Vers des systèmes et outils de notation et de composition pour la musique électroacoustique

Thomas Meyssonnier

université de BORDEAUX

THÈSE PRÉSENTÉE

POUR OBTENIR LE GRADE DE

DOCTEUR DE

L'UNIVERSITÉ DE BORDEAUX

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

SPÉCIALITÉ INFORMATIQUE

Par Thomas MEYSSONNIER

VERS DES SYSTÈMES ET OUTILS DE NOTATION ET DE COMPOSITION POUR LA MUSIQUE ÉLECTROACOUSTIQUE

Sous la direction de Myriam DESAINTE-CATHERINE

Soutenue le 2 novembre 2018

Membres du jury :

Mme. BENOIS-PINEAU Jenny	Professeur	Université de Bordeaux	présidente
M. BRESSON Jean	Chargé de recherche HDR	IRCAM	rapporteur
M. COUPRIE Pierre	Maître de conférence HDR	Sorbonne Université	rapporteur
Mme. SEMAL Catherine	Professeur	Bordeaux INP	examinatrice
M. CADOZ Claude	Ingénieur de recherche	Grenoble INP	examinateur
M. DI SANTO Jean-Louis			invité

Titre : Vers des systèmes et outils de notation et de composition pour la musique électroacoustique

Résumé : Ce travail se situe dans le cadre de la recherche de systèmes de notation permettant de transcrire de façon symbolique l'aspect concret et sensoriel, et non seulement abstrait et structurel, des artefacts de la musique par ordinateur. Dans ce but, nous exposons tout d'abord un modèle formel complet et minimal des objets et structures audionumériques, en référence aux critères de la perception ; ce modèle est implémenté sous la forme d'un langage fonctionnel Turing-potent qui permet d'effectuer l'équivalence entre l'expression mathématique d'un signal et sa réalisation informatique. Puis, nous employons ce formalisme afin d'exprimer un ensemble de critères de synthèse sonore, ce qui donne lieu à un logiciel de synthèse dont l'expressivité est considérable. Ces outils sont organisés suivant le schéma des théories Schaefferiennes, par une décomposition catégorielle dans laquelle les paramètres correspondent à des notions morphologiques. Finalement, nous rendons compte d'une série d'expériences visant à évaluer la pertinence de ces critères dans l'audition humaine, avec le concours d'un musicologue, puis sur un ensemble de sujets, et enfin vis-à-vis d'un public aussi large que possible. Ceci nous conduit à remettre en question la méthodologie la plus adéquate pour traiter ce type de problème, qui nous rapproche des sciences humaines et sociales, et suggère une démarche de science participative.

Mots-clef :

Musique par Ordinateur

Synthèse Sonore

Partitions Virtuelles

Modèle Formel

Théorie Schaefferienne

Psychoacoustique

Title : Towards notation and composition tools and systems for electroacoustic music

Abstract : This piece of work is situated in the context of research on notation systems enabled to transcribe symbolically the concrete and sensorial aspect, and not only the abstract and structural aspect, of computer music artefacts. In this perspective, we first expose a complete and minimal formal model for digital audio objects and structures, relatively to the criteria of perception ; this model is implemented as a Turing-potent functional language, that draws the correspondance between the mathematical expression of a signal and its computational realisation. Then, we apply this formal construction to the expression of a number of schemes for sound synthesis, producing a software synthetiser whose expressivity is consequent. These tools are organised following the lines of Schaefferian theories, through a decomposition into categories whose parameters correspond with morphological notions. Finally, we draw the conclusions of a series of experiments aiming to evaluate the relevance of those schemes in human hearing, with the assistance of a musicologist, then with a number of subjects, and eventually by associating a public that is as wide as possible. This leads us to question the methodology most appropriate to tackle this kind of problem, which brings us closer to social science, and suggests a participative science approach.

Keywords :

Computer Music

Sound Synthesis

Virtual Scores

Formal Model

Schaefferian Theory

Psychoacoustics

Unité de recherche

LaBRI, UMR 5800, 351, cours de la Libération F-33405 Talence cedex

> *When you hear music,*
> *and when it's over,*
> *then it's gone, in the air ;*
> *you can never catch it again...*
>
> – Eric Dolphy

Remerciements

Je tiens à remercier avec une grande sincérité toutes les personnes qui ont participé de près ou de loin au travail que j'ai pu effectuer, ou qui ont contribué d'une façon ou d'une autre à ce que je devienne ce que je suis aujourd'hui.

Tout d'abord Myriam Desainte-Catherine, qui a accepté de me faire confiance, malgré les lourdes contraintes qui pesaient sur mon mode de fonctionnement professionnel, et qui par la suite m'a continuellement soutenu, ce sans quoi rien n'aurait été possible ; également Catherine Semal et Jean-Louis Di Santo, qui non seulement m'ont beaucoup éclairé dans des domaines qui jusque là m'étaient étrangers, mais qui de plus m'ont beaucoup apporté par leur regard à la fois lucide et humain sur la réalité que nous partageons, et par leur attitude rassurante qui a considérablement arrangé mon intégration dans le milieu ; et aussi Thibaud Keller, pour son inépuisable enthousiame et son talent pour trouver une solution à tout problème, ainsi qu'Axel Camus, qui m'a fait l'honneur d'être mon premier étudiant, et qui a eu la patience de le supporter et même de l'apprécier, pour autant que je sache.

Je dois également un grand merci aux personnels administratifs, aux personnels techniques, et aux travailleurs sociaux du LaBRI et du CNRS, et en particulier à Annick Mersier du SCRIME, pour leur soutien inconditionnel sans lequel je n'aurais pas pu travailler dans de bonnes conditions.

Je suis reconnaissant envers les enseignants et les étudiants que j'ai côtoyé durant mes études, et qui m'ont finalement accompagné jusque là où je suis : notamment Claude Cadoz et les enseignants d'AST, sans lesquels il m'aurait été difficile de prendre le virage vers la préoccupation musicale, ainsi que Clément Bossut et Jaime Chao, pour l'engagement envers la musique qu'ils m'ont transmis ; Stéphane Rousset, pour sa façon singulière mais lumineuse de présenter la psychologie cognitive, et Sonia Dupin, dont les multiples talents ne la conduisaient pas à se prendre elle-même au sérieux, ni d'ailleurs ce qui n'en valait pas la peine ; plus anciennement, envers Goeff Finn qui avait déjà pressenti chez moi le penchant artistique, et envers Ross Carson à qui je dois cette fabuleuse ouverture d'esprit qu'est le bilinguisme ; envers Chantal Legros, sans laquelle je n'aurais peut-être jamais trouvé la motivation d'apprendre à jouer de la musique ; envers Gerard Li qui dès l'école primaire m'a montré que la différence était une source de curiosité et d'apprentissage inépuisable qu'il fallait vivre comme une bénédiction ; et finalement envers un grand nombre de personnes qui m'ont aidé à faire les choix que j'ai fait tout au long de mon parcours.

Je dois beaucoup à certaines personnes dont le sens de l'écoute, la bienveillance, la lucidité et le sens des responsabilités m'ont souvent aidé à garder espoir malgré des situations difficiles, notamment Marie-Philippe Decloche, Jean-Michel Clauzel, Pierre Mistral, Monique Escalon et Estelle Joguet ; sans elles et sans eux, la vie m'aurait certainement semblé impossible.

Bien entendu, je ne serais pas le même sans mes amis les plus proches : Pascal & Aurélie Baltazar qui m'ont encouragé à explorer le monde de l'Art, Olivier Tache et James Leonard qui ont été les premiers à reconnaître mon potentiel à faire des choses parfois intéressantes, Gaspard Garcia qui de nombreuses façons et à de nombreuses reprises m'a éclairé, accompagné, fait réfléchir, et qui souvent à exprimé envers moi l'affection dont j'avais besoin, Mehdi Mhalla qui non seulement m'a relancé dans la musique à une époque où j'en avais perdu l'espoir, mais de plus a fait de même en ce qui concerne les études... et beaucoup d'autres, pour lesquels la place me manque [1], mais qui resteront toujours pour moi les personnes avec qui j'ai souhaité faire chemin.

J'ai eu la chance d'être accueilli à Villeneuve (Grenoble), où j'ai pu voir des gens authentiques que la différence n'effraie pas, ainsi que dans le Haut-Diois, qui reste pour moi l'endroit au monde qui ressemble le plus à une terre d'origine ; je remercie donc les habitants de ces lieux pour le sentiment d'appartenance que j'ai reçu de leur part.

Finalement, je ne serais rien sans les membres de ma famille, tous et toutes autant les uns et les unes que les autres, mais plus particulièrement ma mère, pour l'énergie avec laquelle elle m'a soutenu dans les moments les plus difficiles de mon existence, avec une grande opiniâtreté, sans s'arrêter à sa propre souffrance ; mon père (mais cela irait sans dire, étant donné son côté peu cérémonieux), qui m'a transmis un grand nombre de choses, dont l'humilité qui incombe aux chercheurs et un goût prononcé pour le free jazz ; mon frère Charles, avec qui j'ai construit ma personnalité et avec qui je partage tellement de choses qu'il serait difficile d'en faire le tour ; et mon oncle David, grâce auquel j'ai pu déjà très jeune apprendre à trouver du sens dans la musique, à raconter des histoires, et à cultiver l'imaginaire.

1. Mais je peux tout de même citer Ely, Romain, Chris, Pep, Gab, Vince, André, Cyril, Xav, Ben et Marianne, Camila, Christophe, Maud, Anne-Claire, Julie, Delphine, Aurélie, Gaëlle, Mélanie, Roby, Jérôme, Caroline, Alyssa, Micky, Robert, Lorenz et Liseul, Jenny, Clea, Alexis, Anne-Laure, Rémi, Robin, Iléana, Joachim, Danny et Lorenzo, James, Laura, Mathieu, Eric avec qui j'ai passé d'heureuses années à la Cité Scolaire Stendhal, ainsi que Stan, Delphine, Sam, Momo, Cecile, Blandine, Mouss, Thibaut et Maya, Jason, Marina, Céline et toute cette joyeuse bande d'étudiants du CTM ; et encore Marc, Isa, Sam, Jérémie, Fouet et Claire, Xav, Ben et Djoub, ... ainsi que David, qui manque encore après toutes ces années ... et Giss, Elie, Adélaïde, Steph et Luc, Samy, Guillaume, et encore Emilie et Emmanuel, Vince, Ingrid et Bruno, Clem et Clem, Yoann, Olivier, Enzo, Greg et Nat, Maël et Aurélie, Liliane, ... et tellement d'autres dont l'extrême sympathie me suit encore après toutes ces années, qu'ils et elles soient loin ou proches, et y compris si le souvenir m'échappe à l'heure où j'écris ces lignes ... merci !

Table des matières

Chapitre I

Introduction

L'objectif de ce travail est de développer un outil de notation et de composition pour la musique électroacoustique.

En effet, les évolutions technologiques à partir de la deuxième moitié du XX$^{\text{ème}}$siècle ont permis l'émergence de pratiques musicales (dont notamment la *musique concrète*) dans lesquelles le répertoire traditionnel de timbres instrumentaux cédait la place à des sons de nature arbitraire, dont l'origine n'était pas en général identifiable.

Dans ce contexte, les schémas usuels de notation musicale deviennent inopérants, puisqu'il devient nécessaire d'exprimer le concret sonore autant que les structures musicales.

En outre, le contexte d'écoute étant le plus souvent *acousmatique*[1], il devient nécessaire d'analyser le phénomène sonore suivant les critères de la perception, et non selon la façon dont il est produit. Cela conduit de nombreux auteurs, comme Dennis Smalley [25] ou Pierre Schaeffer [24], à s'interesser à une décomposition *spectromorphologique* dans laquelle on distingue la forme temporelle du matériau sonore.

1 Plan de travail

Il existe déjà un certain nombre de systèmes de notation qui prennent en compte les aspects sonores d'une œuvre musicale. Pour n'en citer que quelques uns :

- Le système de Jean-Louis Di-Santo [11], qui propose une notation assortie de quelques extensions pour le « solfège des objets musicaux » de Pierre Schaeffer
- Le système de Lasse Thoresen [28], fondé également sur la théorie Schaefferienne, qui propose une représentation graphique et chronique des éléments d'une œuvre musicale
- Les *unités sémiotiques temporelles* du MIM [20], système fondé sur la notion d'évolution temporelle, indépendamment de ce à quoi elle s'applique
- Le logiciel *EAnalysis* de Pierre Couprie [9], qui permet d'annoter un enregistrement de diverses façons, notamment en référence à des systèmes d'analyse consensuellement établis

Dans tous les cas ces systèmes de notation se réfèrent au phénomène sonore (donc un phénomène de la perception humaine), ce qui entraîne que l'interprétation de l'œuvre analysée ne peut se faire que directement par une intervention humaine. Bien évidemment, effectuer ce processus de façon objective requiert une longue expérience de l'écoute musicale, et il ne peut donc être effectué correctement que par des spécialistes.

1.1 Référence à un modèle audionumérique

Dans le contexte plus spécifique de la musique par ordinateur, il est possible de prendre comme référence pour l'analyse non pas le phénomène sonore lui-même, tel qu'il existe dans la perception humaine, mais le *modèle audionumérique* correspondant à l'œuvre musicale, dont on suppose qu'il est étroitement lié, par l'intermédiaire d'un système de rendu acoustique, au phénomène sonore.

Ainsi on parvient à une possibilité d'objectivation du sonore qui gagne en indépendance par rapport à l'appréciation humaine. Pour faire une analogie, les instruments de musique et la théorie musicale se sont développés de façon conjointe, les instruments pouvant servir de support pour représenter de façon tangible les éléments d'une œuvre musicale, et observer leurs relations. L'ordinateur, en tant que *meta-instrument*[2], peut accomplir le même rôle de concrétisation des éléments d'une œuvre dans le cadre de la musique électroacoustique.

1. C'est à dire que l'origine réelle du son n'est pas apparente, et pas nécessairement pertinente dans son appréciation.
2. C'est à dire que l'ordinateur, par l'intermédiaire de l'ensemble des algorithmes qu'il peut exécuter, peut produire des signaux associés à un large ensemble de timbres sonores.

Un tel système possède notamment un potentiel didactique certain, puisqu'il permet à une personne qui n'a pas nécessairement une grande expérience préalable de rapporter les connaissances musicologiques à l'univers de la synthèse sonore, et ainsi de développer ses compétences d'écoute en se fondant sur un modèle dont l'objectivité est extérieure.

1.2 Mise en œuvre

Afin de pouvoir expérimenter directement avec le système de notation, il est nécessaire que ce modèle audionumérique soit implémenté sous la forme d'un système informatique destiné à la synthèse sonore. Cela permet à l'utilisateur d'apprendre par des allers-retours incessants entre la théorie et la pratique, ce qui, dans les théories modernes de l'apprentissage, constitue un facteur déterminant dans l'appropriation de la connaissance.

On peut alors parler d'*apprentissage actif*, voire même de *connaissances énactives* [29]. Un tel système, si son efficacité est raisonnable, permet d'étayer la culture musicale par des exemples concrets, remettant ainsi l'accent sur l'écoute et non sur des raisonnements théoriques pouvant conduire au non-sens abstrait [3].

Pierre Schaeffer avait pour habitude d'insister sur l'importance d' « entendre ce que l'on fait », car selon lui le repère de l'audition humaine est fondamental dans la pratique musicale ; puisque l'œuvre est destinée à être entendue, elle doit être conçue dès son origine par ce biais. Il est donc indispensable que la notation puisse être interprétée de façon audible, et non seulement comme un assemblage de symboles.

1.3 Analyse des systèmes musicologiques

Moyennant l'existence d'un système théorique et d'un outil logiciel associé pour l'analyse et la composition sonore, nous pouvons alors tenter d'intégrer la *culture* existante de la musique par ordinateur sous la forme de *propriétés mathématiques* et d'*algorithmes informatiques* ; la possibilité d'une telle entreprise constitue un critère fiable d'évaluation de l'expressivité du système, c'est à dire que ses *a priori* ne doivent pas gêner ce processus de transcription.

Cette préoccupation est de nature fondamentale, car il ne s'agit pas de réinventer la musique par ordinateur, mais plutôt de la soumettre à une nouvelle approche d'analyse ; il est possible néanmoins que cette transition dans le domaine informatique nous permette de rendre visibles certains biais présents dans les théories musicologiques, puisqu'étant donnée l'expression d'un schéma culturellement admis, nous pourrons alors voir comment le détourner, voire même le généraliser.

1.4 Confrontation à la réalité de la perception

Une telle analyse nous permet de formuler un grand nombre d'hypothèses sur ce qui constituerait une axiomatisation adéquate de la structure des œuvres musicales ; il devient alors possible d'isoler ces éléments et d'en évaluer l'*universalité dans la perception humaine*, dans un certain contexte culturel ou bien de façon indépendante de celui-ci.

Dans ce but, il est possible, moyennant une *méthodologie adaptée* à la problématique, de nous livrer à des *expérimentations* au carrefour de la psychoacoustique et de la recherche musicale, qui pourront nous renseigner sur la nature des critères perceptivement pertinents pour la qualification des phénomènes musicaux, sur le caractère naturel ou culturel de ces critères, et finalement sur leur existence ou leur non existence.

2 Modèle audionumérique

Au cours du chapitre II page 22, nous posons les bases d'un modèle permettant de décrire de façon complète et flexible les *objets audionumériques*, c'est à dire les artefacts informatiques correspondants à des phénomènes acoustiques et finalement à des phénomènes de la perception sonore.

Le modèle audionumérique le plus commun actuellement est celui de la séquence d'échantillons, qui décrit les variations de la pression acoustique représentée au cours du temps.

Néanmoins ce modèle ne nous convient pas, car ses éléments (les échantillons) ne sont pas individuellement perceptibles d'un point de vue humain. Le modèle que nous proposons fait appel à un ensemble d'éléments de base, ainsi qu'à des règles de construction d'objets complexes, qui sont porteurs de sens dans la perception humaine.

3. Il nous paraît d'ailleurs utile de signaler qu'au cours de ce travail, et grâce à l'outil que nous avons développé, nos compétences d'écoute se sont développées d'une façon remarquable.

En effet, une notation destinée à être *humainement lisible* se doit en premier lieu de représenter le phénomène sonore selon les modalités de la perception humaine du son. Nous commencerons donc par identifier les propriétés qui nous semblent indispensables dans un tel modèle, puis nous déterminerons en conséquence les contraintes qui définissent un formalisme adéquat.

2.1 Propriétés du modèle perceptif

Nous savons à l'heure actuelle finalement très peu de choses sur le processus par lequel les êtres humains parviennent à entendre, c'est à dire à produire les phénomènes de l'esprit correspondants à la structuration psychique du flux acoustique tel qu'il est transmis par les organes externes de la perception.

En voici un résumé très schématique (d'après Brian C.G. Moore [21] ainsi que d'autres source diverses) :

1. La conformation et la distance entre les oreilles externes permet à l'être humain de mesurer, grâce aux différences de volume, de phase et de composition spectrale, la position des sources sonores dans l'espace environnant ; il est donc possible de discerner des objets indépendants dans le flux sonore.
2. Les vibrations acoustiques sont capturées par le biais des tympans, puis transmis à l'oreille interne *via* ces organes minuscules que sont le *marteau* et l'*enclume* ; nous savons donc qu'effectivement, l'être humain est sensible aux variations de la pression acoustique [4].
3. Par suite, la *cochlée* [5], une sorte de tube conique enroulé à la manière d'un escargot, permet du fait de ses propriétés acoustiques de transmettre la vibration au travers de la *membrane basillaire* avec la propriété remarquable de décomposer celle-ci suivant les différentes fréquences élémentaires qui la composent, transposant ainsi l'information dans un domaine spectrotemporel de façon purement physique.
4. Les *cellules ciliées*, ancrées dans la membrane basillaire, assurent ensuite l'interface avec les cellules nerveuses qui permettent l'interiorisation neurobiologique de la perception sonore. Leur fonctionnement est encore largement méconnu, mais il est d'ores et déjà clair que cette transition n'est pas transparente ; par exemple il existe des preuves comme quoi un phénomène de compression [6] est possible à ce stade, afin de minimiser des écarts importants de volume sonore.
5. L'étude des premiers stades de traitement neuronaux de cette information montrent qu'il existe dès le départ plusieurs propriétés spectrales et temporelles distinctes qui sont dégagées d'emblée par les structures de l'oreille interne, ce qui étaye largement l'hypothèse comme quoi il est pertinent de décomposer le phénomène sonore en un certain nombre de catégories qualitativement différentes.

2.1.1 Paradigme de synthèse additive

Les phénomènes de la perception sonore pouvant parfois être considérés comme des objets indépendants superposés, et non comme un flux uniforme, il nous semble pertinent de modéliser les objets audionumériques comme un ensemble d'éléments composés par synthèse additive (c'est à dire par une somme de signaux).

Dans ces conditions, différents éléments seront amenés à fusionner en un seul objet, d'autres formeront éventuellement des *groupes* manifestant un comportement global au sein duquel des individualités restent discernables, et d'autres encore apparaîtront clairement comme des entités sonores complètement indépendantes. Il restera donc à charge de déterminer les conditions dans lesquelles les éléments audionumériques peuvent être regroupés de façon pertinente.

2.1.2 Décomposition spectrotemporelle

Du fait qu'avant tout traitement cérébral, l'oreille elle-même opère une décomposition spectrale du stimulus acoutique, nous procèderons de même à une décomposition spectrotemporelle de la représentation, en éléments de *forme* temporelle et de *matière* spectrale.

La représentation audionumérique sera donc constituée des éléments suivants :

Eléments spectraux fondamentaux : des oscillateurs sinusoïdaux, éventuellement bruités, qui constituent l'aspect matériel du son, et définissent donc son existence ou sa non existence.

4. En fait, d'autres organes comme la cage thoracique peuvent éventuellement accompagner ce mode de perception, ce qui permet notamment à des personnes malentendantes de percevoir au moins les basses fréquences ; mais nous prenons le parti d'ignorer ces phénomènes qui dans le cadre de cette étude sont d'ordre secondaire.

5. La cochlée a également pour fonction notoire de permettre la sensibilité humaine aux changements d'orientation de la tête, ce qui peut dans certaines circonstances interagir avec le processus d'audition ; néanmoins, dans la mesure où notre préoccupation est essentiellement musicale, nous supposerons que l'expérience de l'auditeur est neutre de ce point de vue.

6. En fait, de façon surprenante, il est même possible que les mouvements de ces cellules produisent des vibrations acoustiques réelles dans l'oreille interne, engendrant de ce fait un son indépendant de l'environnement acoustique extérieur.

Eléments temporels fondamentaux : des enveloppes temporelles et des modulateurs pseudo-périodiques ou stochastiques, qui définissent les propriétés d'un élément existant, à une échelle où l'audition humaine est capable de percevoir des variations temporelles.

Ces deux types d'éléments peuvent être décrits par la notion de *fonction temporelle* que nous verrons ci-après, qui constitue un modèle mathématique de *signaux* à fréquence acoustique ou à basse fréquence, la différence entre le spectral et le temporel étant du point de vue audionumérique une simple question d'échelle.

Nous structurerons donc les objets audionumériques à partir de *générateurs de signal* et de *paramètres de contrôle*, ces derniers étant sujets à une évolution temporelle ; en conséquence, le modèle comprendra une notion primordiale de *temporalité*, qui permettra de définir les variations des paramètres au cours du temps, et d'en vérifier la cohérence sémantique.

2.2 Algèbre fonctionnelle

Moyennant la transcription des objets audionumériques sous la forme d'expressions mathématiques, nous pouvons par l'intermédiaire du λ-calcul [6] les formuler dans une algèbre fonctionnelle ad-hoc, fortement inspirée de SCHEME [15] (qui est un dialecte LISP [16], l'implémentation la plus connue du λ-calcul).

En effet, ces formalismes présentent deux propriétés avantageuses, voire même indispensables, vis-à-vis de notre problématique :

- La complétude de leur expressivité, par rapport à ce qui est concevable pour un algorithme, est réputée vraie, et ce de longue date ; étant donné que nous souhaitons éviter d'introduire des biais dans la représentation, il nous semble essentiel de conserver ce potentiel expressif.
- Ces algèbres fonctionnelles sont intentionnellement conçues dans le but de transcrire des expressions mathématiques dans le medium informatique, et sont donc parfaitement adaptées à la tâche qui nous incombe.

2.2.1 Organisation émergente

Le fait d'employer un langage complet, et non simplement un modèle mathématique clos, nous permet d'exploiter les possibilités de structuration offertes par celui-ci. En effet, les langages fonctionnels permettent de traiter les fonctions mathématiques comme des objets linguistiques, et ainsi de travailler à des ordres supérieurs d'abstraction, construits à partir des objets élémentaires.

Cette façon de procéder favorise l'émergence d'un système de catégories, qui regroupe par analogie de structure les objets qui diffèrent seulement par une partie de leur définition. Bien évidemment, le principe même de la notation symbolique repose essentiellement sur la possibilité d'effectuer des regroupement catégoriels, à différents niveaux et suivant différents axes. Il s'agit là d'une propriété fondamentale des langages en général, dont l'importance est primordiale dans le cas qui nous occupe.

En l'occurence, dans le système que nous proposons, ce potentiel à la décomposition en objets élémentaires structurés *a posteriori* est poussé aussi loin que possible, ce qui lui donne une flexibilité dont nous n'avons pas encore vu les limites. En cela il se distingue d'un langage tel que Nyquist [10], qui est comparable sur de nombreux points, mais qui vise une efficacité computationnelle poussée, et à ce titre fonctionne le plus souvent avec des objets plus structurés, qui peuvent bénéficier d'optimisations ; cela introduit néanmoins certains biais dans la représentation, notamment au niveau de la formulation des comportements temporels, qui ont tendance à suivre les conventions de la synthèse sonore associée à des périphériques d'interface instrumentaux.

2.2.2 Langage de haut niveau

Notre but étant de produire une expression humainement lisible des constructions audionumériques, notre système libère l'utilisateur de toute considération de bas niveau, bien que ces préoccupations restent prédominantes en ce qui concerne la synthèse sonore ; en conséquence, les opérations associées doivent être gérées de façon automatique par l'implémentation logicielle. Le langage rend tout cela transparent et ne manipule que des objets élémentaires dont le sens est accessible par la perception humaine, ce qui permet d'interpréter les expressions d'une façon plus naturelle.

Sur ce critère, il diffère donc de CLM/CL [5], qui est un dialecte LISP incorporant des primitives pour la synthèse sonore ; celui-ci offre la possibilité, toujours à des fins d'optimisation, de spécifier manuellement les parties critiques (en temps) des algorithmes dans un environnement spécial qui conduit à une compilation en C++, largement plus efficace que l'interpréteur. Le gain en performance est certainement important, mais cela oblige l'utilisateur à travailler au niveau des échantillons, ce qui n'est pas souhaitable dans notre cas, comme nous l'avons exprimé ci-dessus.

3 Outil pour la synthèse sonore

Dans le chapitre III page 42, nous détaillons le fonctionnement interne de l'implémentation de l'algèbre fonctionnelle que nous définissons ; le langage choisi pour cette réalisation est SuperCollider [26], qui est un langage essentiellement destiné à la synthèse sonore, et qui bénéficie par rapport à de nombreux autres des avancées théoriques modernes telles que la programmation orientée objet et la séparation entre un interpréteur très souple et très expressif, et un serveur dédié à la synthèse sonore extrêmement efficace ; une documentation bien fournie permet d'en exploiter avantageusement les possibilités.

Mis à part le fait que cet outil nous a été d'une aide précieuse dans la suite de notre travail, une des raisons fondamentales de son existence est qu'il agit comme une preuve *de facto* de la possibilité de réaliser une telle construction linguistique ; en effet, le fait qu'il soit possible de traiter automatiquement et d'une façon raisonnablement efficace toutes les préoccupations de bas niveau nous a souvent été contesté, au travers d'un certain nombre d'arguments spécialistes que nous ne citerons pas, et dont d'ailleurs le sens nous échappe, n'ayant pas nous-mêmes cette compétence. En définitive, l'outil existe, ses performances ne sont pas excellentes mais restent assez largement du domaine du pratiquable, et cette piste de développement, en conséquence, nous semble conserver sa validité.

3.1 Choix du langage

La raison pour laquelle nous avons choisi un langage impératif pour la synthèse sonore, plutôt qu'un langage fonctionnel préexistant dans lequel nous aurions pu implémenter des extensions pour cette même synthèse, est finalement assez simple même si elle n'est pas forcément immédiate : implémenter de façon efficace des algorithmes de synthèse sonore constitue une tâche difficile, qui concerne des spécialistes parmi lesquels nous ne nous comptons pas ; en revanche, implémenter une couche fonctionnelle dans un langage qui traite déjà les fonctions comme des objets peut se faire assez simplement, et même si cette implémentation est loin d'être optimale, la perte en performance ne devrait pas avoir un impact significatif sur le résultat puisque la partie critique réside dans les algorithmes de synthèse eux-mêmes.

Néanmoins, il eût été possible d'opter plutôt pour une réalisation employant l'hybridation d'un langage fonctionnel tel que OpenMusic [2] avec un langage spécialisé dans la synthèse sonore tel que FAUST [22] ; cette solution aurait certainement été une des plus élégantes, mais malheureusement à l'époque où le besoin de réaliser cet outil devenait pressant, une documentation suffisante sur FAUST n'était pas disponible [7].

3.2 Gestion des algorithmes de synthèse

Un outil pour la synthèse sonore se doit de gérer d'une façon efficace les parties de son fonctionnement qui traitent directement de la génération de sons ; étant donnée la fréquence minimale acceptable pour produire des enregistrements audionumériques, certaines parties du code devront être exécutées un très grand nombre de fois, en un temps minimal.

La plupart des langages existants [8] imposent à l'utilisateur de formuler des algorithmes de synthèse *invariants*, c'est à dire que le schéma d'exécution doit être défini *a priori* et ne peut plus être modifié en cours d'exécution, ceci afin de pouvoir produire une version optimisée et compilée de ces algorithmes ; SuperCollider ne fait pas exception à cette règle.

Nous parvenons néanmoins à minimiser l'impact de cette contrainte, en utilisant les deux moyens suivants :

- Tout d'abord, les programmes peuvent être écrits comme des assemblages de modules (algorithmes invariants) qui peuvent ensuite être assemblés à loisir par des fonctions qui quant à elles peuvent tout à fait être variantes (par des appels récursifs, ou par des appels à des fonctions passées en argument, par exemple). Cette synthèse *modulaire* repousse assez largement les limites d'une synthèse *monolithique*.
- De plus, une même définition de module peut donner lieu à un grand nombre d'algorithmes similaires, qui ne diffèrent que par les types des valeurs données en entrée ; un système d'inférence du graphe de calcul adéquat est facile à mettre en place, grâce au polymorphisme préexistant dans SuperCollider. Finalement, nous pouvons instancier au besoin les algorithmes individuels qui sont réellement nécessaires dans un programme donné.

7. En revanche, à l'heure actuelle, ce langage est très bien documenté, ce qui suggère qu'une réimplémentation complète serait possible, à supposer que nous en trouvions le temps.

8. En fait, la totalité des langages que nous connaissons ; mais il n'est pas exclu qu'il existe d'autres langages ne présentant pas cette limitation.

3.3 Limitations

La limitation principale de l'outil réside dans le fait qu'il fonctionne essentiellement *hors temps réel*, ce qui convient à la réalisation de partitions mais pas dans le cadre de pratiques d'interprétation en temps réel. Il est envisageable d'étendre ses possibilités afin de le rendre réactif, mais cela suppose (outre l'ajout de primitives adaptées dans l'algèbre) de revoir le mécanisme d'évaluation des expressions, notamment en ce qui concerne la production des instances de modules. Néanmoins, cette perspective reste intéressante et sera étudiée si l'opportunité se présente.

D'autres limitations, comme le peu de documentation disponible, l'incomplétude du mécanisme de gestion d'erreurs, et le fait que le logiciel dépend de SuperCollider (qui est relativement lourd), sont caractéristiques d'un logiciel expérimental et seront potentiellement compensées si il advient qu'il se construise une communauté d'utilisateurs – ce qui ne dépend pas entièrement de nous. Dans l'immédiat, cet outil nous a permis d'avancer quelque peu dans notre travail, ce qui est déjà très satisfaisant.

4 Transcription de la théorie Schaefferienne

Le chapitre IV page 58 expose un cadre général, ainsi qu'une série d'hypothèses, dont le but est de fournir un modèle de synthèse sonore correspondant aux éléments de la théorie musicale mise en avant dans l'œuvre de Pierre Schaeffer [24], dont son disciple Michel Chion a effectué une synthèse remarquable [7].

Parmi l'ensemble des théories existantes, nous avons décidé de nous focaliser sur celle-ci car elle se prête aisément à une interprétation dans les standards de la synthèse sonore ; le fait que Schaeffer a débuté sa carrière en tant qu'ingénieur du son n'y est probablement pas étranger. Quoiqu'il en soit, les notions de décomposition spectrotemporelle, de catégorisation des processus, et de paramètres constants ou variants y sont déjà présentes, ce qui rend la tâche relativement abordable en premier lieu.

Nous proposons ici une brève présentation du fonctionnement de cette théorie, puis nous expliquons comment les critères qu'elle dégage peuvent être transposés dans le domaine des algorithmes de synthèse sonore, au moyen de l'algèbre fonctionnelle que nous avons produite.

4.1 Introduction à la théorie Schaefferienne

L'approche Schaefferienne est une approche *typo-morphologique*, c'est à dire qu'elle tente tout d'abord de façon *qualitative* de séparer le problème en différentes catégories, ce qui donne lieu à un certain nombre de *critères* ; par la suite ces critères peuvent être analysés tour à tour et leurs différents aspects peuvent être traités de façon *quantitative*, en leur affectant des valeurs qui déterminent la *forme* des objets individuels.

Les critères de Schaeffer sont au nombre de sept, par ordre d'importance :

1. Profil mélodique : l'évolution de l'objet dans le champ des hauteurs
2. Dynamique : l'évolution de l'objet en termes d'intensité
3. Masse : la nature de l'objet en termes de matériau sonore – Schaeffer distingue essentiellement les sons *toniques* (doués d'une hauteur distincte) des sons *complexes* (ceux n'ayant qu'une hauteur approximative, comme les bruits ou les sons inharmoniques)
4. Timbre harmonique : à l'interieur de la contrainte de masse, la façon dont l'objet se situe par rapport à d'autres objets analogues, par exemple en termes de *richesse* ou de *couleur* – ce critère est néanmoins considéré comme une continuité
5. Grain : le fait que le matériau sonore puisse donner à entendre des variations à une échelle temporelle microscopique (mais néanmoins audible), ce qui lui donne une texture semblable par analogie à un matériau granuleux
6. Allure : la propriété de l'objet de présenter des variations continues, à une échelle qui donne l'impression qu'il est animé d'un mouvement – Schaeffer qualifie celles-ci de *mécaniques*, *naturelles* ou *vivantes* suivant le caractère régulier, chaotique ou mi-régulier, mi-chaotique du mouvement
7. Profil harmonique : l'évolution du timbre harmonique au cours de la durée d'existence de l'objet

Par la suite, Jean-Louis Di Santo [11] propose d'approfondir la notion de *profil* (l'évolution des paramètres au cours du temps) et de représenter l'objet comme traversant un certain nombre de *phases* temporelles au cours desquelles la variation est homogène (constante ou variant dans le même sens et à une allure régulière). Il ramène également le nombre de critères à quatre, considérés d'importance égale, en exprimant tout ce qui varie potentiellement comme un profil :

- Profil mélodique : essentiellement comme celui de Schaeffer, avec l'ajout des phases qui permet une description plus articulée sur le plan temporel

- Profil dynamique : de même que pour le profil mélodique, le concept est le même avec l'ajout des phases – ceci simplifie grandement le problème de la classification des attaques et des chutes, qui dans Schaeffer donnait lieu à une sous-partie conséquente du système
- Profil de masse : à la fois la masse, subdivisée en six catégories (*tonique*, *bruit*, *inharmonique*, *tonique-bruit*, *tonique-inharmonique*, *inharmonique-bruit*), et le timbre et le profil harmonique, considérés comme faisant partie intégrante de la masse
- Profil rythmique : cette notion regroupe les notions de grain et d'allure, qui sont perçues comme des variations rythmiques de l'objet ; elle inclut également les propriétés itératives des objets, qui dans Schaeffer sont représentées par des catégories d'objets séparées

Parmi les critiques pouvant être formulées à l'encontre de la théorie Schaefferienne, nous pouvons citer une tendance observable à répondre à l'influence de la technique du son et des modèles mathématiques des signaux, ainsi que l'*a priori* fondamental comme quoi la perception sonore est composée d'objets distincts. Effectivement, ces deux points sont complètement discutables d'un point de vue philosophique, mais ils conditionnent également les réalisations informatiques de toutes sortes, du fait de la nature même des ordinateurs ; en conséquence, nous retiendrons ces limitations comme faisant partie intégrante de la rationalité scientifique occidentale, que nous ne sommes pas en mesure de remettre en question dans le cadre de cette étude.

4.2 Analyse des critères

Nous retenons les critères de Jean-Louis Di Santo, ainsi que sa notion de temporalité ; nous rangeons ces critères dans deux grandes catégories, les critères de *forme* d'une part, et les critères de *matière* d'autre part. Du point de vue de la synthèse sonore, ceci semble correspondre à la distinction entre les éléments de génération spectrale, et les paramètres temporels qui déterminent leur évolution.

4.2.1 Forme

Nous assimilons la notion de profil à l'évolution d'un paramètre au cours du temps ; un même objet peut admettre un grand nombre de paramètres, outre sa hauteur et son intensité, qui sont indispensables. Les valeurs de paramètres, lorsqu'elles ne sont pas constantes, sont représentées par des *enveloppes* dont la valeur parcourt un certain nombre de segments au cours du temps.

Seuls les paramètres sont affectés par la temporalité de l'objet (le contenu spectral, quant à lui, se situe à une échelle temporelle bien inférieure) ; en ajoutant l'hypothèse comme quoi un objet possède une cohérence temporelle dans laquelle les processus qui le définissent sont en quelque sorte synchronisés, cela nous conduit à représenter la temporalité comme un ensemble de phases communes à tous les paramètres de l'objet.

L'aspect rythmique du comportement de l'objet, quant à lui, est représenté par des *modulateurs* périodiques, pseudo-périodiques ou stochastiques qui modifient d'une certaine façon la valeur de certains paramètres. Ces modulateurs sont eux-mêmes définis par d'autres paramètres, qui déterminent leur fréquence et l'amplitude de la modulation.

4.2.2 Matière

En ce qui concerne le contenu spectral de l'objet, qui dans notre modèle est engendré par un ensemble d'oscillateurs, nous souhaitons distinguer deux aspects fondamentaux de ce type de système :

- la *structure spectrale*, c'est à dire la façon dont les composantes sont agencées dans le domaine fréquentiel
- l'*enveloppe spectrale*, c'est à dire l'allure de la courbe qui à chaque fréquence associe l'amplitude de l'éventuelle composante concernée

Alors que l'enveloppe spectrale pourrait à la rigueur être considérée comme un élément au sein d'un continuum [9], la structure spectrale se situe clairement parmi un ensemble de différents types qui définissent des catégories disjointes. Dans la mesure du possible, nous tentons de nous ramener à une situation où les variations continues restent possibles, de façon à ne pas introduire de limites artificielles dans la formulation de l'algorithme ; ce principe est mis en œuvre par le fait que les différents éléments algorithmiques qui modifient la structure ou l'enveloppe spectrale sont chacun munis d'au moins un paramètre qui peut en neutraliser l'effet.

Il est possible, malgré le grand nombre d'éléments algorithmiques proposés, de conserver une efficacité raisonnable dans la tâche de synthèse, du fait que lorsqu'un élément est absent du début à la fin de l'objet, le

9. Il faut néanmoins prendre en compte que celle-ci doit être décrite au moyen d'une fonction analytique, et qu'il n'existe pas d'expression générale en intention de l'ensemble desdites fonctions analytiques.

paradigme de synthèse modulaire permet d'éliminer tout simplement le module associé dans la réalisation de cet objet [10].

Nous notons au passage que l'enveloppe spectrale peut se référer soit à la fréquence absolue des composantes de l'objet (ce que nous appelons la *couleur*), soit à la fréquence des composantes relativement à une fréquence de référence de l'objet (éventuellement dynamique), que nous appelons *fréquence de base*.

4.3 Démarche empirique

La détermination des éléments de ce modèle ne peut se faire de façon purement déductive et rationaliste, puisqu'il est fondamental au cours de ce processus de reposer autant que possible sur l'appréciation humaine de la qualité des sons.

Nous adoptons donc une démarche inductive et empirique, au moyen du système de synthèse sonore que nous avons produit, et dirigé par la perception qui en résulte. Ainsi il est possible, à partir des éléments de base de la notation, d'identifier l'organisation catégorielle des différentes familles de structures complexes ; ce travail est nécessairement entrepris avec la participation active de spécialistes de l'analyse musicale, dont l'expertise sert de référence.

5 Evaluation des hypothèses

Finalement, le chapitre V page 73 présente les tentatives expérimentales que nous avons mises en œuvre afin d'évaluer la validité des hypothèses que nous avons pu formuler concernant la perception humaine des sons de synthèse : existe-t'il de telles catégories, y a t'il des seuils mesurables des basculements perceptifs suggérés ?

Ces expériences sont au nombre de trois, la première avec la participation de Jean-Louis Di Santo (musicologue et compositeur), la seconde auprès d'un ensemble de volontaires disponibles dans nos réseaux, et la troisième par le biais d'une plateforme Web ouverte à un public aussi large que possible.

Néanmoins l'aspect le plus important de cette démarche réside dans la façon dont elle nous a conduit à questionner la nature même de la recherche musicale, et à proposer des pistes dans le but de définir des protocoles expérimentaux susceptibles de nous renseigner dans ce domaine.

5.1 Méthodologie

La validation de la correspondance entre les critères de synthèse sonore et des catégories bien définies de la perception humaine des sons nécessite de procéder d'une façon qui laisse le moins de champ possible à des interprétations erronnées ; l'expérimentation sur des sujets humains ayant systématiquement lieu dans un contexte difficile, voire impossible à maîtriser ne serait-ce que dans les grandes lignes, une telle entreprise nécessite tout d'abord d'examiner avec soin toutes les questions qu'il nous est possible de concevoir, ayant trait aux risques d'erreur. En voici quelques unes :

- La notion de *critère perceptif* doit tout d'abord être bien définie. En effet, le fait qu'un phénomène soit perceptible, ou qu'une variation dans ce phénomène engendre une variation observable dans la perception, constitue un prérequis nécessaire, mais à notre sens insuffisant. Si nous souhaitons pouvoir affirmer qu'il existe dans la perception une différence catégorielle entre deux phénomènes, suivant leurs paramètres d'instance au sein d'une classe, il nous faut démontrer l'existence d'un *seuil* au niveau duquel la perception bascule de façon radicale sur de petites variations paramétriques ; de plus, affirmer l'universalité de cette distinction catégorielle suppose d'observer que ce seuil est plus ou moins le même, quel que soit l'individu. En outre, au-delà de son existence, la mesure précise d'un seuil est une procédure laborieuse et qui demande des moyens considérables.
- La notion d'*universalité* des observations est elle aussi profondément discutable ; quoiqu'il en soit une expérience n'est effectuée qu'auprès d'une fraction infime de l'humanité, qui doit pourtant être choisie de façon à être représentative dans la mesure du possible. Deux facteurs interviennent en premier lieu : le *nombre* de personnes impliquées, bien entendu, mais également – et cela est beaucoup plus difficilement quantifiable – la diversité présente dans ce groupe de personnes, sur autant de critères que possible, soient-ils physiques, psychiques, sociaux ou culturels. Il va sans dire qu'il est d'ordinaire assez difficile de former un groupe de sujets qui n'est pas entièrement contenu dans un certain milieu social ; en l'occurence, les personnes présentant un intérêt pour la musique, et ayant la compétence de la percevoir d'une façon claire, constituent d'une certaine façon un milieu social à part entière.

10. A titre expérimental, et bien que cela puisse se révéler coûteux en termes de ressources, il serait néanmoins intéressant de tenter de produire des objets très longs, possédant un grand nombre d'éléments modificateurs, voire tous, qui seraient activés de façon intermittente au cours de la durée de vie de l'objet ; par ce biais il est envisageable d'obtenir ce que Schaeffer dénommait une *musique plastique*, c'est à dire une œuvre se résumant à l'évolution du matériau d'un seul objet de longue durée.

- Toujours sur cette notion d'universalité, il est relativement évident dans ces conditions que la culture constitue un biais de premier ordre, en particulier étant donné que la musique est une chose profondément culturelle, comme le démontre l'observation des pratiques musicales dans des cultures autres que la nôtre. Alors, distinguer ce qui serait réellement *naturel*, c'est à dire présent chez n'importe quel être humain, indépendamment de la culture et de l'époque, de ce qui est en fait *culturel*, c'est à dire le produit des apprentissages contextuels des individus (mais qui peuvent s'appliquer à un grand nombre d'individus distincts), peut constituer un problème proprement insurmontable [11].
- Un autre problème bien connu en psychoacoustique est celui de l'interaction possible entre différents critères : il est possible d'observer un certain résultat avec un certain type de sons (par exemple des sons dont la hauteur se situe systématiquement à 440Hz), sans pour autant que ce résultat puisse être reproduit avec un ensemble de sons différents mais présentant le même potentiel d'observation (dans notre exemple, des sons de hauteur aléatoire). Bien entendu, nous ne pouvons pas savoir *a priori* quels autres critères sont susceptibles d'interagir avec celui que nous souhaitons observer, même si nous pouvons à tout le moins tenter d'anticiper les problèmes les plus évidents.

Quoiqu'il en soit, il est notoire dans le milieu des sciences cognitives qu'une expérience comporte toujours des biais et des facteurs non contrôlés qui ont échappé à la vigilance de ses organisateurs ; nous pouvons seulement espérer, après avoir fait tout notre possible, que ces désordres protocolaires ne modifieront pas de façon significative le résultat de l'expérience. En règle générale, les résultats ne peuvent néanmoins être considérés comme dignes de confiance que lorsqu'ils ont été reproduits plusieurs fois par des équipes indépendantes et dans des contextes différents.

5.2 Impératifs pratiques

A ces difficultés inhérentes à la pratique expérimentale sur des sujets humains (ou d'ailleurs sur toute forme de vie), viennent s'ajouter d'autres difficultés au regard des conditions matérielles dans lesquelles se déroulent une expérience. En règle générale, l'organisation d'un tel processus est relativement exigeante en termes de ressources, et il est nécessaire de trouver un compromis pratiquable entre ce qui serait nécessaire à un résultat solide et ce qui est possible étant donnés les moyens disponibles :

- Ne serait-ce qu'au niveau du matériel nécessaire pour un rendu sonore uniforme (ce qui a certainement un impact sur les performances des sujets), il faut soit disposer d'une installation permanente (et donc d'un local et de personnes pour le faire fonctionner), soit s'en remettre à un service réseau ; dans le deuxième cas, le matériel employé par les participants, et les réglages associés, peuvent produire des résultats très inégaux, ce qui peut modifier les observations.
- Les moyens humains affectés à un tel projet peuvent prendre des proportions importantes : combien de personnes sont nécessaires, combien de temps devront-elles y passer, quel niveau d'expertise est attendu de leur part ? Le plus souvent il est difficile de trouver ne serait-ce qu'une quantité minimale de personnes acceptant de s'investir ; il est possible de proposer aux sujets une rémunération, mais bien sûr cela s'avère le plus souvent extrêmement coûteux.
- Finalement, le temps total nécessaire à la conduite du projet expérimental peut lui-même être un facteur bloquant, étant donnée l'échelle sur laquelle on se situe. La recherche scientifique étant ponctuée d'échéances plus ou moins distantes, il est tout à fait possible qu'un projet doive se terminer alors qu'une expérience est en cours, avec des conséquences souvent néfastes.

5.3 Démarche de recherche

Devant la difficulté de mettre en place une démarche expérimentale, au moins pour les raisons que nous avons données ci-dessus, il nous a semblé important de rechercher un équilibre entre les objectifs scientifiques poursuivis et la réalité concrète de la recherche musicale.

Tout d'abord, il nous paraît primordial de bien définir le but visé, tout en restant dans les limites de ce qui peut raisonnablement être espéré, que ce soit pour des raisons théoriques ou purement pratiques. Il serait inutile de perdre un temps considérable dans une tentative se révélant *a posteriori* vouée à l'échec. Il est certainement concevable d'établir la preuve d'un grand nombre de propositions scientifiques, mais en pratique les opportunités sont suffisamment rares pour nous conduire à nous concentrer sur l'essentiel.

Deuxièmement, une méthodologie adéquate devrait impérativement prendre en compte la différence réelle entre la psychoacoustique et la recherche musicale. Dans notre cas, il n'est pas nécessairement souhaitable de rechercher

11. A titre d'exemple, il serait probablement difficile de trouver dans ce monde et à cette époque une personne qui ignore complètement ce qu'est une cannette de Coca-Cola. Et pourtant, nous ne pouvons pas conclure que cette image constitue une catégorie naturelle de la perception humaine, bien qu'étant dans l'incapacité de produire un contre-exemple.

des résultats formellement indéniables, mais le plus souvent, plutôt, d'alimenter des hypothèses pouvant servir de ressources à la création artistique, qui quant à elle n'a pas besoin de preuves pour avancer.

Finalement, la recherche d'une vérité scientifique naturelle et immuable n'est peut-être pas prioritaire, dans la mesure où la culture de la musique par ordinateur est en constante évolution, au moins à notre époque. Nos objectifs pourraient peut-être résider dans la recherche de vérités contextuelles, culturelles et chroniques, qui pourraient accompagner ces mutations et donner un éclairage rationnel au développement de ces pratiques, sans en attendre la conclusion.

Chapitre II

Modélisation des objets audionumériques

Nous souhaitons tout d'abord pouvoir décrire le phénomène étudié – c'est à dire le phénomène sonore, du point de vue de la perception humaine – d'une façon uniforme, et formellement cohérente. Bien entendu, il nous est impossible de manipuler par l'informatique ni les phénomènes de la perception, ni même les phénomènes acoustiques qui le plus souvent en sont à l'origine. En conséquence, nous nous contenterons d'une représentation des phénomènes informatiques permettant la synthèse de sons acoustiques, susceptibles d'être entendus ; nous nommons ces phénomènes *objets audionumériques*.

Nous proposons donc un langage informatique, fondé sur un formalisme mathématique, qui décrit de façon claire la nature des processus algorithmiques permettant d'effectuer la synthèse sonore des objets de notre investigation.

La première étape (1, 2, 3) consiste à donner une expression mathématique du phénomène acoustique, représenté par un *signal* (c'est à dire une pression acoustique variant au cours du temps). Cette expression peut ensuite (4, 5) être transcrite dans l'algèbre fonctionnelle que nous avons présentée précédemment [17]. L'implémentation de l'algèbre permettra ensuite de produire des algorithmes pour la synthèse sonore, pouvant être effectivement réalisés au moyen d'un ordinateur.

Un exemple de mise en œuvre du langage, et du formalisme associé, est donné en 6.

1 Modèle mathématique de la temporalité

On donne ici des définitions d'objets mathématiques auxquels on se réfère plus tard, dans le but de transcrire de façon formelle la notion d'*objet audionumérique* (c'est à dire un objet informatique dont l'interprétation sémantique correspond à un phénomène acoustique, ou au phénomène sonore associé dans la perception humaine).

Autant que possible, les objets seront décrits en intension (notamment par des fonctions) et non en extension (par des séquences de constantes, comme les séquences d'échantillons par exemple).

1.1 Fonction algébrique

Par *fonction algébrique*, on entend toute fonction de $(\mathbb{N}\, ou\, \mathbb{R})^k, k \in \mathbb{N}$ dans $\mathbb{N}$ ou $\mathbb{R}$ définie à partir des opérateurs arithmétiques standard et des fonctions usuelles. On s'intéressera surtout à des fonctions continues de $\mathbb{R}$ dans $\mathbb{R}$.

1.2 Fonction temporelle

Les *fonctions temporelles* sont des fonctions dont la valeur dépend du temps (ou pour être plus exact, de la variable par laquelle on réalise l'abstraction temporelle). On donne ici le détail de leurs propriétés.

A noter que les termes *signal* et *objet audionumérique*, utilisés plus spécifiquement pour les représentations audionumériques des sons audibles, désignent également des fonctions temporelles.

1.2.1 Base de temps

La *base de temps* $\mathbb{T}$ est un sous-ensemble de $\mathbb{R}$ qui sert de support aux fonctions temporelles. $\mathbb{T}$ peut être par exemple $\mathbb{Z}$, $\mathbb{Q}$ ou $\mathbb{R}$. Il est impératif que $\mathbb{T}$ ait une structure de groupe additif, étant donné qu'on souhaitera notamment effectuer des différences. Par contre la base de temps est définie à facteur d'échelle près (elle pourrait

être $\delta \cdot \mathbb{Z}$, $\delta \in \mathbb{R}^*$), et donc une structure d'anneau n'est pas nécessaire (on ne devrait pas avoir besoin d'effectuer un produit de deux valeurs temporelles).

Les fonctions temporelles sont systématiquement définies au moins sur $\mathbb{T}$ ou un de ses sous-ensembles (intervalle dans le cas continu ou séquence dans le cas discret), avec éventuellement d'autres arguments de types divers. Leur valeur est numérique, en général dans $\mathbb{R}$.

Le symbole t sera employé par la suite pour désigner la *variable temporelle* avec la propriété implicite $t \in \mathbb{T}$.

1.2.2 Echantillonnage

On raisonne dans le cas où $\mathbb{T}$ est $\mathbb{R}$, ce qui permet une spécification mathématique plus élégante ; on pourra ensuite restreindre $\mathbb{T}$ à une *base de temps échantillonnée* (discrète) $(\mathbb{T}_\delta = \delta \cdot \mathbb{Z}, \delta \in \mathbb{R}) \subset \mathbb{T}$, de façon à rendre praticable le calcul systématique des valeurs des fonctions temporelles ; δ est alors la *période d'échantillonnage*. Cela entraînera nécessairement les limitations usuelles qui sont exprimées par le théorème d'échantillonnage de Nyquist-Shannon [27] (notamment la limitation de la bande passante à $\frac{1}{2 \cdot \delta} Hz$, et la nécessité de filtrer les signaux par un passe-bas de cette fréquence pour éliminer les discontinuités).

Cet échantillonnage fonctionne par substitution des fonctions temporelles, par la fonctionnnelle ech_δ, qui est définie de la façon suivante, où $\mathcal{E}$ est un ensemble quelconque :

$$ech_\delta : \left(f : \begin{cases} \mathbb{T} \to \mathcal{E} \\ t \mapsto f(t) \end{cases} \mapsto \left(ech_\delta(f) : \begin{cases} \mathbb{T} \to \mathcal{E} \\ t \mapsto f(t_\delta \in \mathbb{T}_\delta \subset \mathbb{T}),\ t_\delta \, t.q. \, t \in [t_\delta, t_\delta + \delta[\end{cases} \right) \right)$$

En pratique, les signaux de fréquence audible et les fonctions temporelles à basse fréquence (qu'on nommera *contrôles*) pourront être échantillonés sur des bases différentes, de façon à réduire la charge de calcul lorsqu'une haute résolution temporelle n'est pas nécessaire. Il suffit pour cela de choisir des *périodes d'échantillonnage* δ_A (pour les signaux) et δ_K (pour les contrôles) telles que $\exists k \in \mathbb{N} \, t.q. \, \delta_K = k \cdot \delta_A$; alors $\mathbb{T}_{\delta_K} \subset \mathbb{T}_{\delta_A}$ et k est ce qu'on appelle usuellement la *taille de bloc* (et pour des raisons pratiques c'est le plus souvent une puissance de deux).

Il est bien sûr théoriquement possible d'utiliser un nombre arbitraire de bases d'échantillonnage, si elles sont récursivement incluses les unes dans les autres ; en pratique les langages de synthèse audionumérique n'en proposent que deux (pour les signaux et pour les contrôles), donc l'implémentation du modèle devra se conformer à cette contrainte.

1.3 Support temporel et signature temporelle

Nous nous intéressons maintenant à la façon dont il est possible de spécifier la temporalité d'un ensemble d'objets décrits par des fonctions temporelles pouvant se référer les unes aux autres (de façon causale, c'est à dire sans références cycliques).

Pour cela nous ferons appel aux notions de *support temporel* (le domaine de définition d'une fonction temporelle) et de *signature temporelle* (l'ensemble des temps internes au support qui interviennent dans l'expression de cette fonction).

1.3.1 Support temporel

Le *support temporel* $T = [T_0, T_f] \subset \mathbb{T}$ d'une fonction temporelle $x(t)$ est défini de la façon suivante :

- $x(t)$ est défini pour tout $t \in T$
- toute fonction temporelle y de support T' dont dépend l'expression de x doit être définie pour tout $t \in T$; en d'autres termes, nécessairement $T \subseteq T'$.

1.3.2 Cohérence temporelle

Ces hypothèses permettent de vérifier la *cohérence temporelle* d'un assemblage d'objets, c'est à dire la propriété qui garantit que les valeurs des fonctions temporelles sont systématiquement choisies à l'intérieur du domaine de définition de ces fontions.

On peut éventuellement dans ce but réaliser l'*extension* d'une fonction temporelle x de support $T = [T_0, T_f]$ vers une fonction x' de support $T' = [T'_0, T'_f]$, telle que $T \subseteq T'$. Dans ce cas :

$$x'(t) = \begin{cases} x(t),\ \forall t \in T \\ 0,\ \forall t \in (T' \setminus T) \end{cases}$$

Si x est une fonction continue et si $x(T_0) = x(T_f) = 0$, alors x' est également continue. Ceci permet notamment de définir par sommation (cf. 1.4) un signal à partir d'un ensemble de signaux définis sur des sous-supports inclus, en en réalisant les extensions au support englobant.

1.3.3 Signature temporelle

Une *signature temporelle* définie sur un support $T = [T_0, T_f]$ est une séquence $(T_l)_{0 \leq l \leq L} \in \mathbb{T}$ qui à un indice entier l tel que $0 \leq l \leq L$ fait correspondre un *point-clef* $T_l \in \mathbb{T}$. $L \in \mathbb{N}$ est la *longueur* de cette signature temporelle (en termes de nombre de segments). Cette séquence doit respecter les hypothèses suivantes :

- $T_L = T_f$
- $\forall i, j \in \{0, .., L\}$, $i < j \Leftrightarrow T_i \leq T_j$

On notera simplement $(T_l)_L$ et $\{(T_l)_L\}$ respectivement la signature temporelle de longueur L et l'ensemble des signatures temporelles de longueur L, bien qu'en réalité ils correspondent respectivement à $(T_l)_{0 \leq l \leq L}$ et $\{(T_l)_{0 \leq l \leq L} | \forall l \in \{0, .., L\}, T_l \in \mathbb{T}\}$.

Cette notion correspond au concept de *phase* qui apparaît dans le développement de la théorie Schaefferienne présenté par Jean-Louis Di Santo [11]. Le principe consiste à décomposer l'évolution temporelle des objets en intervalles dans lesquels les variations sont homogènes.

En conséquence, lorsque les fonctions temporelles qui définissent un objet font référence à une date interne à son support temporel, cette date sera exprimée comme un élément de signature. Ceci permet d'une part, d'exprimer des notions de synchronicité entre objets par le partage d'une même signature, et d'autre part, de formuler des familles d'objets en faisant abstraction de la quantification temporelle ; on parle alors de *morphologie non quantifiée* ou de *morphologie abstraite*, voir 3.1.

1.4 Opérations algébriques sur les fonctions temporelles

Il est possible d'appliquer des opérations algébriques sur les fonctions temporelles, de la façon suivante :

Si on a la fonction algébrique $f : \begin{cases} \mathbb{R}^k \to \mathbb{R} \\ x_1, ..., x_k \mapsto f(x_1, ..., x_k) \end{cases}$, on peut la transposer dans le domaine temporel par la fonction $f^{\#} : \begin{cases} (\mathbb{T} \to \mathbb{R})^k \to (\mathbb{T} \to \mathbb{R}) \\ x_1, ..., x_k \mapsto f^{\#}(x_1, ..., x_k) \end{cases}$

telle que $\forall t$, $f^{\#}(x_1, ..., x_k)(t) = f(x_1(t), ..., x_k(t))$.

Le typage [1] des éléments supprimant l'ambiguïté entre les nombres et les fonctions temporelles, cette substitution peut être effectuée de façon implicite et on représente donc ce type d'opération algébrique avec les mêmes opérateurs.

1.5 Fonctionnelles temporelles

Il existe deux types de *fonctionnelles temporelles* (c'est à dire des fonctions temporelles d'ordre supérieur) :

- les *fonctionnelles temporelles à contrôle dynamique*, dont les arguments peuvent être d'autres fonctions temporelles
- certains *profils*, qui sont les membres des types engendrés par l'*opérateur de profilage*. Celui-ci permet d'extraire un type dynamique (temporel) d'un type quelconque (le plus souvent statique)

1.5.1 Fonctionnelle temporelle à contrôle dynamique

Les fonctionnelles temporelles à contrôle dynamique sont des objets de type :

$(\mathbb{T} \to \mathbb{R})^k \times \mathcal{X} \to \mathcal{Y}$, *avec* $k \in \mathbb{N}$ *et* $\mathcal{X}, \mathcal{Y}$ *deux types quelconques*

Comme on le verra plus loin (cf 4.3.3), c'est le cas notamment de certains générateurs de signaux (audibles ou de contrôle) dont certains paramètres sont dynamiques (et sont donc des fonctions temporelles).

Le terme de *filtre*, plus spécifiquement, désigne les fonctionnelles temporelles qui associent un signal audible à (entre autres) un ou plusieurs signaux audibles.

1.5.2 Opérateur de profilage

Dans certains cas on souhaite substituer à un nombre ou une fonction (le plus souvent indépendants du temps) un objet analogue mais qui dépend du temps. On parle alors du *profil* de l'objet originel, qui représente son

1. Pour mémoire, la notion de *type* couvre un ensemble employé comme ensemble de départ ou d'arrivée d'une fonction ; on appelle la séquence de types en argument, et le type de retour, la *signature* de cette fonction. Mais en général ici, le mot *signature* se réfère plutôt à la signature temporelle définie en 1.3.

évolution dynamique ; par exemple, une enveloppe temporelle correspond au profil d'une valeur constante au cours du temps.

On introduit pour cela l'*opérateur de profilage*, qui est un opérateur sur les types :

Soit un type $\mathcal{X}$ *quelconque*

Alors on note $\mathcal{X}^P$ *le type* $\mathbb{T} \to \mathcal{X}$

On dit que $\mathcal{X}^P$ est l'*ensemble des profils* de $\mathcal{X}$.

Si on a $x \in \mathcal{X}$, on notera pour plus de clarté un *profil* associé $x^P \in \mathcal{X}^P$. Cette notation est purement symbolique et n'entraîne aucune conséquence formelle, l'opérateur de profilage s'appliquant uniquement à des types. Le risque d'ambiguïté lorsqu'on applique l'opérateur de profilage à un type de types n'est pas pertinent dans la suite.

Ce procédé permet notamment dans la suite de substituer $x^P(t)$, $x^P \in \mathcal{X}^P$ à $x \in \mathcal{X}$ pour obtenir des comportements dynamiques à partir d'un ensemble de comportements statiques.

Si $\mathcal{X}$ est un type de fonctions, alors $\mathcal{X}^P$ est un type de fonctionnelles temporelles. Dans le cas contraire, $\mathcal{X}^P$ est simplement un type de fonctions temporelles.

1.6 Fonctions temporelles aléatoires

Les *fonctions aléatoires* $Rand_X|_a^b(t)_\delta$ sont des fonctions temporelles telles que pour tout symbole X :

$$\forall k \in \mathbb{N},\ si\ t \in [k \cdot \delta, (k+1) \cdot \delta]\ alors :$$
$$Rand_X|_a^b(t)_\delta\ est\ une\ variable\ aléatoire\ distribuée\ uniformément\ sur\ [a, b]$$

Une autre définition est possible (afin d'obtenir une fonction continue) :

$$Soit\ (p_n)\ une\ suite\ de\ variables\ aléatoires\ distribuées\ uniformément\ sur\ [a, b]$$
$$\forall k \in \mathbb{N},\ si\ t \in [k \cdot \delta, (k+1) \cdot \delta]\ alors :$$
$$CRand_X|_a^b(t)_\delta = (\frac{t}{\delta} - k) \cdot p_k + (k + 1 - \frac{t}{\delta}) \cdot p_{k+1}$$

Dans la suite on notera plus simplement $Rand_X|_a^b(t)$ et $CRand_X|_a^b(t)$ les fonctions $Rand_X|_a^b(t)_\delta$ et $CRand_X|_a^b(t)_\delta$, avec δ fixé à une valeur suffisamment faible pour obtenir un bruitage satisfaisant (voir 3.2.1 et 3.2.2).

Autrement dit, les fonctions aléatoires correspondent à la temporalisation d'une réalisation quelconque d'un *processus aléatoire* (ceux-ci sont décrits avec plus de détails dans [14]). Bien que ce soit théoriquement faux, nous prendrons pour hypothèse que dans le contexte dans lequel elles sont employées, toutes les réalisations d'un processus aléatoire de cette catégorie sont équivalentes du point de vue psychoacoustique ; en effet, il existe des réalisations aberrantes mais elles sont suffisamment rares pour pouvoir être négligées en pratique.

2 Objectivation des phénomènes audionumériques

Le formalisme, comme tout système de notation, opère une *objectivation* des phénomènes ; c'est à dire qu'il manipule des objets statiques (des algorithmes) afin de représenter des phénomènes dynamiques (les signaux). A un signal donné (une fonction temporelle), on fait donc correspondre un objet qui n'a pas le temps pour variable (un algorithme de génération, qui est invariant au cours du temps) ; on effectue donc une *curryfication* de façon à obtenir un *objet temporel* X à partir de la fonction x que l'on souhaite modéliser :

$$\blacksquare\ A\ x : \begin{cases} \mathcal{G} \times \mathbb{T} \to \mathbb{R} \\ G, t \mapsto x(G, t) \end{cases} \quad on\ associe\ X : \begin{cases} \mathcal{G} \to (\mathbb{T} \to \mathbb{R}) \\ G \mapsto (t \mapsto x(G, t)) \end{cases} \quad , pour\ \mathcal{G}\ ensemble\ quelconque$$

Dans le cas des profils qui peuvent être contenus dans l'expression, ce procédé se déroule en deux temps : le type $(\mathcal{X} \to \mathcal{Y})^P = \mathbb{T} \to (\mathcal{X} \to \mathcal{Y})$ devient tout d'abord $\mathbb{T} \times \mathcal{X} \to \mathcal{Y}$ par *retrocurryfication*, et peut alors être curryfié en $\mathcal{X} \to (\mathbb{T} \to \mathcal{Y})$.

3 Notion de temporalité

La notion de *temporalité* est une notion transversale, qui d'une manière générale désigne l'ensemble des informations d'ordre temporel qui définissent les constructions mathématiques employées ici.

3.1 Valeurs temporelles et objets temporels

Le principe de la *morphologie non quantifiée* consiste à spécifier les fonctions temporelles de façon abstraite (qui se réfère à des valeurs non numériques), puis d'employer des opérations permettant de situer ces fonctions dans une temporalité *réalisable* (dans le sens où elle ne contient que des dates numériques, à l'exception de tout autre type de valeur).

Après avoir exposé comment des structures temporelles de ce type peuvent être exprimées, nous verrons de quelle façon elles peuvent être manipulées (3.2).

3.1.1 Valeurs temporelles abstraites

Une *mesure temporelle* est une fonction $\tau \in \mathcal{V}'$ définie de la façon suivante :

$\tau : \mathbb{T} \to \mathbb{T}$

Au cours des opérations sur les valeurs temporelles, l'argument passé à τ sera systématiquement la durée du support ou du segment de signature associé, ce qui permet d'exprimer des durées relatives à celui-ci ; par exemple $t \mapsto t$ représente cette durée elle-même. On qualifie ce type de spécification de temps *abstrait*.

On utilise également des fonctions constantes du type $t \mapsto k \in \mathbb{T}$ pour exprimer des temps constants. Etant donné qu'une fonction constante peut être considérée comme équivalente à une constante, il est possible de définir concrètement un objet défini par ce type de spécification. On parlera alors de temps *réalisable*.

Ce procédé permet d'exprimer les dates qui caractérisent un objet audionumérique de façon à ce qu'elles puissent être interprétées *a posteriori*, en leur appliquant des distorsions temporelles arbitraires.

Il est possible d'effectuer des opérations sur τ, avec les définitions suivantes :

$$a, b \in \{\mathbb{T} \to \mathbb{T}\} \implies a(b) \in \{\mathbb{T} \to \mathbb{T}\} : (t \mapsto a(b(t))) = a \circ b$$

$$a \in \mathbb{T}, b \in \{\mathbb{T} \to \mathbb{T}\}, f \in \{\mathbb{T}^2 \to \mathbb{T}\} \implies \begin{cases} f(a,b) \in \{\mathbb{T} \to \mathbb{T}\} : t \mapsto f(a, b(t)) \\ f(b,a) \in \{\mathbb{T} \to \mathbb{T}\} : t \mapsto f(b(t), a) \end{cases}$$

$$a, b \in \{\mathbb{T} \to \mathbb{T}\}, f \in \{\mathbb{T}^2 \to \mathbb{T}\} \implies f(a,b) \in \{\mathbb{T} \to \mathbb{T}\} : t \mapsto f(a(t), b(t))$$

3.1.2 Valeurs temporelles indexées

Une *valeur temporelle abstraite* $v \in (\mathcal{V} = \mathcal{V}' \cup \mathcal{V}'_S)$ peut être soit une simple mesure temporelle $\tau \in \mathcal{V}'$, soit une *valeur temporelle indexée*, c'est à dire un couple $(n, \tau) \in (\mathcal{V}'_S = \mathbb{N} \times \mathcal{V}')$ où $n \in \mathbb{N}$ et $\tau \in \mathcal{V}'$ est une mesure temporelle. Ce deuxième cas permet d'exprimer des dates par rapport aux indices d'une *signature temporelle abstraite* (c'est à dire une séquence $(T_l)_L \, t.q. \forall l \in \{0, ..., L\} : T_l \in \mathcal{V}'$). Ce type de valeur $v \in \mathcal{V}'_S$ n'est pas réalisable, même si sa mesure est constante.

Un *support temporel abstrait* est alors un intervalle entre deux valeurs temporelles abstraites, et une éventuelle séquence de points-clef est une séquence de valeurs temporelles abstraites. La morphologie non quantifiée permet de spécifier des objets fondamentaux dont le support abstrait est $[t \mapsto 0, t \mapsto t]$ et la séquence éventuelle de points-clef $\{t \mapsto 0, (1, t \mapsto 0), ..., (L-1, t \mapsto 0), t \mapsto t\}$ (pour un objet à L segments). Ces deux notions pourront ensuite être ramenées respectivement à un support temporel ou à une signature temporelle tels qu'ils sont définis en 1.3 page 23.

3.1.3 Réalisation des valeurs temporelles abstraites

Une temporalité est dite réalisable si toutes ses valeurs temporelles sont non indexées et de mesure constante. On pourra alors effectuer la substitution suivante, sur toutes les valeurs temporelles :

La *réalisation* $\tau^R \in \mathbb{T}$ d'une valeur temporelle $\tau \in \mathcal{V}'$ est :

$\exists T \in \mathbb{T} : \forall t \in \mathbb{T} : \tau(t) = T \Rightarrow \tau^R = T$

3.1.4 Objets temporels

Un *objet temporel* $O \in \mathcal{O}$, de *réalisation* $O^R \in (\mathbb{T} \to \mathbb{R})$, est un triplet *(fonction, paramètres, temporalité)* $(F, P, T) \in \mathcal{F} \times \mathcal{P} \times \mathcal{T}$ avec les définitions suivantes :

Les paramètres $P \in \mathcal{P} = \bigcup_{p,q \in \mathbb{N}} \mathcal{P}_{p,q}$ sont $(x_1, ..., x_p, o_1, ..., o_q) \in \mathcal{P}_{p,q} = (\mathbb{R}^p \times \mathcal{O}^q)$.

La temporalité $T \in \mathcal{T} = \mathcal{T}' \cup \mathcal{T}'_L$ est soit $(T_0, T_f) \in \mathcal{T}' = \mathcal{V}^2$, soit $(T_0, T_f, (T_1, ..., T_{L-1})) \in \mathcal{T}'_L = (\mathcal{V}^2 \times \mathcal{V}^{L-1})$.

La fonction F est définie de la façon suivante :

$$F : \begin{cases} \begin{cases} \mathcal{P}_{p,q} \times \mathcal{T}' & \to \big(\mathbb{T} \to \mathbb{R}\big) \\ P, T & \mapsto \Big(t \in [T_0^R, T_f^R] \mapsto F\big(x_1, ..., x_p, o_1^R(t), ..., o_q^R(t), T_0^R, T_f^R\big)\Big) \end{cases} \\ OU \\ \begin{cases} \mathcal{P}_{p,q} \times \mathcal{T}'_L & \to \big(\mathbb{T} \to \mathbb{R}\big) \\ P, T & \mapsto \Big(t \in [T_0^R, T_f^R] \mapsto F\big(x_1, ..., x_p, o_1^R(t), ..., o_q^R(t), T_0^R, T_f^R, \{T_0^R, ..., T_{L-1}^R, T_f^R\}\big)\Big) \end{cases} \end{cases}$$

Alors la *réalisation* O^R de O est :

$O^R : t \mapsto O^R(t) = F(P, T)(t)$

Celle-ci n'est définie que lorsque les valeurs temporelles qui définissent O et ses sous-objets sont toutes réalisables.

Ce système permet de spécifier un ensemble d'objets élémentaires de morphologie non quantifiée, puis par l'application d'*opérations temporelles* (3.2) de les projeter dans une temporalité réalisable et donc d'obtenir un objet audionumérique.

3.1.5 Opérations algébriques sur les objets

Il est possible d'étendre les opérations algébriques usuelles (somme, produit, etc...) afin qu'elles puissent également être appliquées à des objets, de la façon suivante :

Soit $\star$ *un opérateur algébrique quelconque, alors :*

$\star : \mathcal{O}^2 \to \mathcal{O}$ *est défini comme :*

$$(o_1, o_2) \mapsto \begin{cases} \Big(F : \big((o'_1, o'_2), (T_0, T_f)\big) \mapsto \\ \Big(t \in [T_0^R, T_f^R] \mapsto \star\big(o_1'^R(t), o_2'^R(t)\big)\Big), \\ P = (o_1, o_2), T = (t \mapsto 0, t \mapsto t)\Big) \end{cases}$$

Ceci peut s'appppliquer à n'importe quel opérateur mathématique binaire, ainsi qu'à l'extension de ceux-ci à des opérateurs n-aires par associativité. Dans ce qui suit on ne fera pas de distinction entre les opérations algébriques appliquées à des nombres, et celles appliquées à des objets temporels.

3.2 Opérations temporelles sur les objets

Le concept d'*opération temporelle* représente l'application, sur les valeurs temporelles qui définissent un objet ainsi que ses sous-objets, d'une fonction $f_T : \mathcal{V} \to \mathcal{V}$ qui effectue une transformation quelconque sur ces valeurs temporelles.

Pour décrire ce concept, nous emploierons une fonctionnelle $map_T : (\mathcal{V} \to \mathcal{V}) \to (\mathcal{O} \to \mathcal{O})$, qui à une fonction de transformation des valeurs temporelles, associe l'application de cette fonction aux valeurs temporelles qui définissent un objet temporel.

Une *opération temporelle* est donc une fonction $map_T(f_T) : \mathcal{O} \to \mathcal{O}$ définie de la façon suivante :

$$map_T : \begin{cases} (\mathcal{V} \to \mathcal{V}) & \to (\mathcal{O} \to \mathcal{O}) \\ f_T & \mapsto \left[\begin{array}{l} \Big(\Big(F, (x_1, \ldots, x_p, o_1, \ldots, o_p), \big(T_0, T_f, (T_1, \ldots, T_{L-1})\big)\Big) \mapsto \\ \Big(F, \big(x_1, \ldots, x_p, map_T(f_T)(o_1), \ldots, map_T(f_T)(o_p)\big), \\ \big(f_T(T_0), f_T(T_f), (f_T(T_1), \ldots, f_T(T_{L-1}))\big)\Big)\Big) \end{array}\right] \end{cases}$$

Ceci signifie que $map_T(f_T)(O)$ applique la fonction f_T à toutes les valeurs temporelles T_l qui définissent l'objet O, et s'applique récursivement à tous les sous-objets o_i qui définissent O.

Des exemples d'opérations temporelles sont donnés en 6 page 35.

3.2.1 Extension du support temporel

Le concept d'*extension* d'un support temporel $[a^R, b^R]$ vers un support englobant $[t_0^R, t_f^R] \supseteq [a^R, b^R]$ (cf. 1.3.2) peut être modélisé par la fonction $ext|_{t_0}^{t_f} : \mathcal{O} \to \mathcal{O}$, dont la définition est la suivante :

$$ext|_{t_0}^{t_f} : \Big(F : P, T \mapsto \big(t \in [a^R, b^R] \mapsto F(P, T)(t)\big), P, T\Big)$$

$$\mapsto \Big(F' : P, T \mapsto \begin{cases} t \in [a^R, b^R] & \mapsto F(P, T)(t) \\ t \in [t_0^R, a^R] \cup [b^R, t_f^R] & \mapsto 0 \end{cases}, P, T\Big)$$

Ceci signifie que $ext|_{t_0}^{t_f}(o)^R(t)$ est $o^R(t)$ si $t \in [a^R, b^R]$, et 0 si $t \in [t_0^R, t_f^R] \setminus [a^R, b^R]$. Cette opération permet donc de définir un objet à partir d'objets définis sur des sous-supports, sans engendrer d'incohérence du point de vue des domaines de définition temporels.

3.2.2 Projection dans un support

L'opérateur $sup|_{t_0}^{t_f}$ permet de projeter le support temporel abstrait d'un objet x dans $[t_0, t_f]$, où t_0 et t_f sont des mesures temporelles quelconques. Toutes les valeurs temporelles abstraites de x sont transformées de la façon suivante :

$sup|_{t_0}^{t_f} = map_T\Big(\tau \in \mathcal{V}' \mapsto \tau(t_f - t_0) + t_0\Big) \circ ext|_{t_0}^{t_f}$

Il est nécessaire d'effectuer au préalable l'extension de l'objet vers le nouveau support, car il est possible qu'il soit défini sur un support plus petit.

Les objets à valeurs temporelles indexées n'appartiennent pas au domaine de définition de *sup*. Si t_0 et t_f sont toutes deux des fonctions constantes, alors l'objet résultant est réalisable.

Cet opérateur permet de spécifier les dates de début et de fin d'un objet temporel, soit de façon absolue soit relativement au résultat d'autres opérations temporelles effectuées *a posteriori*.

Un exemple est donné en 6.4.2 page 39.

3.2.3 Projection dans une signature

L'opérateur $sign_{(T_l)_L}$ permet d'appliquer la signature abstraite $(T_l)_L$ à un objet x, ce qui a pour effet de transformer ses valeurs temporelles indexées en valeurs temporelles simples (c'est à dire non indexées). Les valeurs temporelles simples sont inchangées. Son opération est la suivante, sur toutes les valeurs temporelles abstraites de x :

$$sign_{(T_l)_L} = map_T\left(\begin{cases}\tau \in \mathcal{V}' & \mapsto \tau \\ (n, \tau) \in \mathcal{V}'_S & \mapsto \tau(T_{n+1} - T_n) + T_n\end{cases}\right)$$

Les T_l peuvent être des mesures temporelles quelconques (constantes ou non). Cet opérateur étant le seul qui soit défini sur les objets à valeurs temporelles indexées, il doit nécessairement apparaître directement au dessus de tels objets, si on souhaite transformer leur temporalité en temporalité réalisable.

De même, cet opérateur permet d'affecter la date des points-clef d'un objet temporel. Là encore, cette affectation peut résulter en des dates absolues ou relative à des opérations temporelles ultérieures.

Un exemple est donné en 6.2.3 page 37.

3.2.4 Agrégation de supports temporels adjacents

L'opérateur *seq* permet, étant donnée une séquence de n objets $(x_i)_{n-1}$ de supports abstraits $[\alpha_0, \beta_0]$, $[\alpha_1, \beta_1]$, ..., $[\alpha_{n-1}, \beta_{n-1}]$, de l'agréger en un seul objet de support $[(0, \alpha_0), (n-1, \beta_{n-1})]$, par opération sur l'ensemble des valeurs temporelles abstraites des objets x_i contenus dans ces supports. Pour toute valeur temporelle abstraite de tout objet x_i défini sur $[\alpha_i, \beta_i]$, $i \in \{0, ..., n-1\}$, l'opération est la suivante :

$seq_i = map_T\Big(\tau \in \mathcal{V}' \mapsto (i, \tau)\Big)$

On peut alors définir *seq* de la façon suivante :

$$seq : (x_i)_{n-1} \mapsto \left[\begin{array}{l}\Big(F : \big((o_0, \ldots, o_{n-1}), (T_0, T_n, (T_1, \ldots, T_{n-1}))\big) \mapsto \\ \Big(\forall i \in \{0, \ldots, n-1\} : t \in [T_i^R, T_{i+1}^R] \mapsto seq_i(o_i)^R(t)\Big), \\ P = (x_0, \ldots, x_{n-1}), T = \Big(t \mapsto 0, t \mapsto t, \big((1, t \mapsto 0), \ldots, (n-1, t \mapsto 0)\big)\Big)\Big)\end{array}\right]$$

Les objets à valeurs indexées sont en dehors du domaine de définition de *seq*, et étant donné que celui-ci produit des objets à valeurs indexées, il doit apparaître en dessous d'un *sign* si on souhaite obtenir des objets réalisables.

Cet opérateur permet de composer un support temporel plus grand (et toujours continu comme il se doit) à partir de support temporels indépendants, en les mettant bout à bout. Il est donc employé pour décrire les objets composés d'une séquence d'objets plus petits.

Un exemple est donné en 6.2.2 page 35.

3.2.5 Superposition

L'opérateur *par* permet de superposer un tuple d'objets $(o_1, \ldots, o_N)$ afin d'obtenir leur somme, définie sur le plus petit support qui contiennent tous les leurs. Si on pose :

$$o_i = \Big(F : P, T \mapsto \big(t \in [a_i^R, b_i^R] \mapsto F(P,T)(t)\big), P, T\Big)$$

$T_{min} = min(\{a_1^R, \ldots, a_N^R\})$

$T_{max} = max(\{b_1^R, \ldots, b_N^R\})$

alors on peut définir cet opérateur de la façon suivante :

$$par(o_1, \ldots, o_N) = \begin{cases} \Big(F : (o_1, \ldots, o_N, T_0, T_f) \mapsto \\ \big(\forall t \in [T_0, T_f] : t \mapsto \sum_{i=1}^{N} ext|_{T_{min}}^{T_{max}} o_i^R\big), \\ P = (o_1, \ldots, o_N), T = (T_{min}, T_{max})\Big) \end{cases}$$

Cet opérateur permet donc de décrire des objets composites, dont les composantes sont un ensemble d'objets sur lesquels il n'existe en général aucune hypothèse de synchronicité.

Un exemple est donné en 6.3.3 page 38.

3.2.6 Retard

L'opérateur de retard ret_θ permet de situer un objet x de temporalité réalisable dans une temporalité plus générale, avec un retard $\theta \in \mathbb{T}$ relativement à un temps de référence 0. Pour toutes les valeurs temporelles de x, l'opération est la suivante :

$$ret_\theta = map_T\Big(\tau \in \mathcal{V}' \mapsto \tau + \theta\Big)$$

Cet opérateur n'est défini que pour les objets réalisables (dont les valeurs temporelles sont non indexées et de mesure constante).

Il permet de situer les objets les uns par rapport aux autres dans le référentiel global d'une structure temporelle ; il est possible de cumuler plusieurs opérateurs de retard, ce qui permet de décomposer une structure de façon hiérarchique, en blocs dont la temporalité relative peut être organisée indépendamment de sa situation globale.

Un exemple est donné en 6.4.3 page 40.

4 Primitives pour la construction des objets

On s'interesse ici aux primitives nécessaires à la construction des objets temporels. Ces composantes sont de plusieurs types différents, du plus simple au plus complexe :

- nombres (entiers et réels)
- séquences de nombres
- fonctions algébriques
- objets temporels

On donne ci-dessous les définitions mathématiques de ces constructeurs (lorsqu'elle n'est pas triviale), ainsi que leur expression dans l'algèbre fonctionnelle FLSC (*Functional Language for Sound Composition*) que nous avons définie précédemment [17], et que nous allons utiliser par la suite. Ce langage est un dialecte fonctionnel dont le comportement est semblable à celui de SCHEME [15], et qui possède des extensions qui rendent possible la synthèse sonore. Une présentation complète de SCHEME est disponible en référence [12].

4.1 Nombres et fonctions algébriques

Les constantes numériques sont des éléments de $\mathbb{N}$ ou $\mathbb{R}$. Elles sont représentées en algèbre fonctionnelle par des chaînes littérales numériques.

Les fonctions et opérateurs mathématiques usuels (arithméthiques, trigonométriques, logarithmiques, etc...) peuvent être appliqués à des nombres dans la limite de leur domaine de définition. Ils sont représentés par des fonctions prédéfinies (`+`, `*`, `sin`, `log`,...).

Des fonctions algébriques du type $\mathbb{R}^k \to \mathbb{R}$ peuvent être construites au moyen de la forme spéciale `lambda`, tout comme en SCHEME.

4.2 Séquences

Les séquences sont de la forme $(a_0, a_1, ..., a_N), N \in \mathbb{N}$. Leurs éléments a_i peuvent être de types variés, une séquence peut être hétérogène. Les séquences sont construites au moyen de l'opérateur `list`.

Les opérateurs employés pour construire les séquences, accéder à leurs éléments, les parcourir et effectuer sur elles des opérations usuelles (concaténation, extraction d'une sous-séquence, etc...) sont les opérateurs standards des langages fonctionnels tels que LISP ou SCHEME (`cons`, `car`, `cdr`, `mapcar`, `append`,...).

4.3 Objets temporels

Les fonctions temporelles du type $\mathbb{T} \to \mathbb{R}$ sont construites à partir d'un ensemble de primitives spécifiques du dialecte fonctionnel que nous avons défini [17]. Ces primitives sont de quatre types différents :

- fonctions constantes
- fonctions d'enveloppe
- fonctions pseudo-periodiques
- générateur élémentaire de signal

Il est possible à partir de ces fonctions de construire des objets temporels, qui sont les éléments fondamentaux d'un graphe de calcul audionumérique.

L'objet temporel associé à la fonction f sera noté f^O, mais il s'agit simplement d'une convention de notation qui n'entraîne aucune conséquence sémantique.

4.3.1 Fonctions constantes

Le cas le plus simple est celui où on souhaite définir une fonction temporelle dont la valeur est constante (c'est à dire x telle que $\forall t,\ x(t) = \alpha \in \mathbb{R}$). Puisque la notion de type exclut toute ambiguïté entre les valeurs numériques et les fonctions temporelles, ce cas sera traité par conversion implicite : les valeurs numériques sont prises comme des constantes temporelles lorsqu'une fonction temporelle est attendue. Les fonctions constantes sont donc représentées de la même façon que les nombres.

4.3.2 Fonctions d'enveloppe

La notion Schaefferienne de *profil* (mélodique, dynamique, etc...) décrit l'évolution d'un critère au cours du temps. Les objets Schaefferiens n'étant pas de durée infinie, il est possible de modéliser ces comportements au moyen de fonctions temporelles qui passent par interpolation au travers d'une séquence de points-clef.

La fonction env_L est définie à partir des valeurs temporelles de la signature temporelle $(T_l)_L$ de l'objet et d'une séquence $(a_l)_L$ des $L+1$ niveaux associés. Sa valeur dépend de la fonction d'interpolation $interp(\alpha)$ employée, qui doit associer $[0,1]$ à $[0,1]$, 0 à 0 et 1 à 1 (et de préférence être croissante).

La valeur de env_L se définit de la façon suivante :

$\forall t \in [T_0, T_L]\, t.q.\, T_l \leq t \leq T_{l+1} : env_{L,interp}((a_l)_L, t) = a_l + interp\left(\frac{t-T_l}{T_{l+1}-T_l}\right) \cdot (a_{l+1} - a_l)$

Pour des raisons de simplicité, on n'utilise ici que l'interpolation linéaire ($interp = id$), mais dans le cas général de l'interpolation par segments *interp* peut être définie de façon variée (voire même dépendre de l comme par exemple dans l'interpolation par branches cubiques).

De nombreux résultats peuvent être atteints par interpolation linéaire, en appliquant si nécessaire une opération algébrique à la valeur de l'enveloppe.

Nous pouvons modéliser les enveloppes à L segments linéaires comme une famille d'objets temporels :

$$env_L^O(\alpha_0, \ldots, \alpha_L) = \begin{cases} \Big(\Big(F : (a_0, \ldots, a_L, \ldots, T_0, \ldots T_L) \mapsto \\ t \in [T_l^R, T_{l+1}^R] \mapsto a_l + \frac{t-T_l^R}{T_{l+1}^R - T_l^R} \cdot (a_{l+1} - a_l)\Big), \\ P = (\alpha_0, \ldots, \alpha_L), T = \Big(t \mapsto 0, t \mapsto t, \big((1, t \mapsto 0), \ldots, (L-1, t \mapsto 0)\big)\Big)\Big) \end{cases}$$

Ces enveloppes sont construites au moyen de l'opérateur `env` :

```
(env levels)
```

`levels: séquence de nombres`

Elles peuvent être définies indépendamment de la signature temporelle des objets qui les contiennent ; il est donc possible que la même enveloppe prenne un sens différent, suivant les opérations temporelles dont elle est l'objet.

4.3.3 Fonctions pseudo-périodiques

De nombreux objets contiennent des modulations périodiques ou *pseudo-périodiques* (qui correspondent à la variation dynamique des paramètres d'une fonction périodique).

D'une manière générale, les modulateurs périodiques sont définis comme suit :

$\forall t \in [T_0, T_f] : mod_x(f, \phi, t) = x\Big(\int_{T_0}^{t} f(\tau)d\tau + \phi \Big)$, où x est une fonction périodique au sens usuel du terme (sinus, onde carrée, triangulaire, en dents-de-scie, etc...).

Si f est constante, le modulateur est périodique. Autrement il est pseudo-périodique, sachant que f peut être définie à partir d'enveloppes ou d'autres modulateurs.

L'objet temporel associé est défini de la façon suivante :

$$mod_x^O(f, \phi) = \begin{cases} \Big(F : (\phi, f, T_0, T_f) \mapsto \\ \Big(t \in [T_0^R, T_f^R] \mapsto x\big(\int_{T_0}^{t} f^R(\tau) \cdot d\tau + \phi \big) \Big), \\ P = (\phi, f), T = (t \mapsto 0, t \mapsto t) \Big) \end{cases}$$

Les objets de ce type sont construites par des opérateurs spécifiques dont la signature est la même :

```
(mod freq phi)
mod: un opérateur pseudo-périodique
freq: une fonction temporelle
phi: un nombre
```

Les opérateurs pseudo-periodiques seront par exemple `lfo` (sinusoïde), `squ` (carré), `tri` (triangle), etc...

4.3.4 Générateur élémentaire de signal

Les composantes élémentaires d'un signal sont produites par des oscillateurs sinusoïdaux, définis comme $osc = mod_{sin}$ (cf. Fonctions pseudo-périodiques et 1.6).

Les objets temporels associés $osc^O = mod_{sin}^O$ sont construits par l'opérateur `osc`, défini de la même manière :

```
(osc freq phi)
freq: une fonction temporelle
phi: un nombre
```

A la différence des fonctions pseudo-périodiques, le générateur élémentaire de signal produit un signal audio et non un signal de contrôle. C'est la seule primitive qui soit dans ce cas, un objet audionumérique ne peut donc être produit que par la composition de ces générateurs de signal.

Un exemple est donné en 6.3.1 page 37.

4.3.5 Fonctions stochastiques

La génération de composantes bruitées nécessite d'effectuer une modulation stochastique de la fréquence, afin d'obtenir un bruit à bande limitée [14]. Ce type de modulateur est ce que l'on nomme une fonction stochastique.

Leur expression est celle des fonctions *Rand* et *CRand* que nous avons donnée en 1.6.

Les objets temporels associés sont définis de la façon suivante, pour $XRand = Rand$ ou $XRand = CRand$:

$$XRand^O(a, b, \delta) = \begin{cases} \Big(F : (a, b, \delta, T_0, T_f) \mapsto \\ \Big(t \in [T_0^R, T_f^R] \mapsto XRand_X|_a^b(t)_\delta, \quad , \textit{où } X \textit{ est unique} \\ P = (\phi, f), T = (t \mapsto 0, t \mapsto t) \Big) \end{cases}$$

Ces objets sont construits par les opérateurs `rand` et `crand` :

```
(rand max min freq)
(crand max min freq)
max, min: deux nombres
freq: une fonction temporelle
```

Chaque instance de fonction stochastique produit une fonction différente ; il est possible néanmoins de réutiliser la même instance plusieurs fois au moyen de la forme spéciale `let`.

5 Formes spécifiques à la synthèse sonore

5.1 Modules

Le but de l'évaluation d'un programme complet est de produire un *fragment audionumérique*, c'est à dire une suite d'échantillons dans la base de temps discrète souhaitée.

Un fragment peut résulter de l'agrégation d'un ensemble de fragments, par sommation de leur extension vers le support temporel le plus petit qui contienne l'ensemble des supports d'origine.

Les fragments élémentaires sont obtenus par appel à des *modules*, qui sont un type spécifique de fonction. Certains modules produisent des signaux audio (qui peuvent être interprétés comme des fragments, si leur temporalité est réalisable), d'autres produisent des signaux de contrôle (qui peuvent alors servir de paramètres pour d'autres modules).

5.1.1 Forme module

La forme `module` permet de définir des modules, c'est à dire des éléments optimisés de graphe de calcul audionumérique. Elle définit donc un environnement optimisé qui encapsule une combinaison de primitives des objets temporels (cf. 4.3).

La syntaxe est la suivante :

```
(module (parms) fonct)
```

`parms: une séquence de paramètres`

`fonct: une fonction temporelle`

Cette forme a donc la même signature que `lambda`, et se comporte de façon analogue sauf que :

- Elle retourne un module, qui est forcément défini par un objet temporel (contrôle ou audio).
- Afin de garantir la possibilité de l'optimisation (ce qui correspond à une compilation préalable), elle doit être *algorithmiquement invariante*, c'est à dire que son graphe de calcul doit être statiquement défini à sa création. Cette forme ne peut donc pas prendre de fonction en argument ; ses paramètres `parms` sont des scalaires ou d'autres modules.
- Les paramètres de type séquence sont interprétés comme la composition parallèle d'un ensemble de modules produisant des signaux audio.

5.1.2 Instanciation

Les modules définis par `module` (ainsi que les modules prédéfinis) peuvent être instanciés par un appel de fonction ordinaire. Ceux-ci constituent la base de l'expression qui définit un graphe de calcul, puisqu'ils permettent d'exprimer les objets temporels (audio et contrôle) qui constituent le fragment audionumérique qu'il produit.

La temporalité des modules est définie selon le principe de la morphologie non quantifiée (cf. 3), et peut ensuite être transformée par des opérations temporelles (cf. 5.2) afin d'obtenir des objets réalisables.

Le résultat d'un appel est une instance de module, qui peut être passée en paramètre à une autre instance de module, ou si il produit un fragment, être intégré au résultat du programme tout entier.

5.1.3 Modules prédéfinis

En pratique il est possible de fonctionner uniquement avec des modules élémentaires prédéfinis, il n'est donc pas nécessaire de faire appel à `module`. On fera correspondre aux primitives `osc`, `lfo`, `squ`, `tri`, `rand`, `crand` leurs équivalents modulaires `mosc`, `mlfo`, `msqu`, `mtri`, `mrand`, `mcrand` qui ont exactement la même syntaxe.

Les enveloppes constituent une exception puisque dans une perspective modulaire, il est plus judicieux de les composer séquentiellement (cf. 5.3) à partir de segments élémentaires, ce qui permet d'obtenir un nombre variable de segments.

Un segment linéaire est défini de la façon suivante :

$seg(a,b,t) : \forall t \in [T_0, T_f], t \mapsto a + \frac{t-T_0}{T_f-T_0} \cdot (b-a)$

L'objet temporel associé est donc :

$$seg^O(a,b) = \begin{cases} \Big(\big(F : (a,b,T_0,T_f) \mapsto \\ \big(t \in [T_0^R, T_f^R] \mapsto a + \frac{t-T_0^R}{T_f^R-T_0^R} \cdot (b-a)\big)\Big), \\ P = (a,b), T = (t \mapsto 0, t \mapsto t) \Big) \end{cases}$$

En lieu et place de `env` on utilisera `mseg`, qui a la syntaxe suivante :

```
(mseg start end)
start: un scalaire
end: un scalaire
```

Les paramètres `start` et `end` correspondent respectivement au niveau de début et de fin du segment. La temporalité du segment est quant à elle définie comme le début et la fin du support temporel qui sera appliqué sur l'objet.

Un exemple est donné en 6.2.1 page 35.

On utilisera également des modules prédéfinis pour les opérations arithmétiques standard : `m+`, `m*`, `m-`, `m/`, etc...

5.2 Altérations de la temporalité

La spécification des modules fait abstraction de la quantification de leur temporalité. Celle-ci est précisée *a posteriori* au moyen des fonctions d'altération de la temporalité, qui permettent les opérations suivantes :

Retard : permet de positionner les fragments dans la temporalité de la partition

Définition de support : permet de délimiter le support temporel sur lequel un sous-graphe sera défini

Définition de signature : permet d'appliquer une signature temporelle sur un sous-graphe défini par rapport à une signature abstraite

5.2.1 Fonction de retard

Le positionnement des fragments dans la temporalité globale de la partition nécessite de faire appel à un opérateur de retard qui sera réalisé par la forme `delay`. Cette forme produit un fragment audionumérique à partir d'un fragment existant, elle peut donc y être substituée partout où cela est pertinent.

Sa sémantique est celle de l'opération temporelle de retard ret_θ (cf. 3.2.6), c'est à dire qu'elle applique un retard $\theta \in \mathbb{T}$ à l'ensemble des valeurs temporelles qui définissent un objet.

Un nombre arbitraire de `delay` pourront être appliqués les uns par dessus les autres, ce qui aura pour effet de cumuler leur temps caractéristiques.

La syntaxe fonctionnelle de `delay` est la suivante :

```
(delay time frag)
time: un temps
frag: un fragment
```

Un exemple est donné en 6.4.3 page 40.

5.2.2 Définition de support

La limitation temporelle d'un sous-graphe peut être réalisée au moyen de la fonction `base`, qui projette ce sous-graphe sur le support spécifié.

Sa sémantique est celle de l'opération $sup|_{t_0}^{t_f}$ (cf. 3.2.2), c'est à dire que le sous-graphe est temporellement limité à l'intervalle $[t_0, t_f]$.

La syntaxe fonctionnelle de `base` est la suivante :

```
(base pair sig)
pair: une paire de temps
sig: un signal audio
```

La paire `pair` est une liste de deux temps qui correspondent respectivement au début et à la fin du nouveau support.

En raison de l'hypothèse de cohérence temporelle, cette fonction ne peut s'appliquer qu'à un signal audio ; les signaux de contrôle doivent nécessairement être définis sur le même support que celui des objets qui en dépendent.

Dans le cas des signaux audio, leur extension vers un support plus grand est réalisée de façon implicite, lorsque c'est nécessaire.

Le résultat peut constituer un fragment lorsque la temporalité résultante est réalisable, ce qui se produit lorsque les bornes du support sont des nombres.

Un exemple est donné en 6.4.2 page 39.

5.2.3 Définition de signature

Lorsque la définition d'un objet se réfère à des points-clef abstraits, il est nécessaire de préciser ceux-ci au moyen de la fonction `sign`, dont la sémantique est celle de l'opération $sign_{(T_l)_L}$ (cf. 3.2.3) ; les points-clef sont affectés aux indices successifs de la signature spécifiée.

La syntaxe fonctionnelle de `sign` est la suivante :

```
(sign list sig)
```

```
list: une liste de temps
```

```
sig: un signal
```

Etant donné que cette fonction ne modifie pas le support, la liste `list` ne contient que les éléments internes de la signature (les indices $i \in \{1, \ldots, L-1\}$).

Un exemple est donné en 6.2.3 page 37.

5.3 Opérateurs de composition

Les opérateurs de composition permettent d'effectuer des combinaisons de modules autres que les combinaisons arithmétiques standard ou la dépendance fonctionnelle d'un module sur un autre, qui sont forcément synchrones.

Il s'agit ici de pouvoir moduler le support temporel et la signature temporelle auxquels on fait référence. Cette question se résout différemment suivant que l'on compose des signaux audio ou de contrôle, pour la raison suivante :

Lorsqu'on compose des signaux de contrôle, le résultat du calcul est employé par un autre module. En conséquence, le support temporel de la composition doit être le même que celui du module de destination. En outre les opérateurs existants permettent déjà d'effecuter des combinaisons synchrones. En conséquence, la combinaison des contrôles est nécessairement une composition séquentielle, qui se traduit par un partionnement du support temporel qui produit une séquence de sous-supports juxtaposés.

Lorsqu'on compose des signaux audio, en revanche, ceux-ci peuvent se superposer partiellement, ou inversement laisser des trous. On parle alors par opposition de composition parallèle, c'est à dire qu'un ensemble de signaux audio est mis en superposition. Il est possible de définir pour chacun de ces signaux un support temporel distinct qui sera étendu vers celui de la composition, ce qui permet des assemblages non synchrones.

5.3.1 Composition séquentielle

L'opérateur de composition séquentielle s'applique aux contrôles et effectue un partitionnement du support temporel de base, selon une signature abstraite.

Sa sémantique est celle de *seq* (cf. 3.2.4).

Sa syntaxe est la suivante :

```
(seq mods)
```

```
mods: une liste de signaux
```

La liste `mods` est la séquence des modules qui correspondent aux différentes parties de cette partition. Son nombre d'éléments détermine le nombre de segments de la signature abstraite à laquelle la compostion se réfère. Le $i^{\text{ème}}$ module aura un support borné entre les indices $i-1$ et i de la signature qui sera appliquée.

Le résultat de `seq` est du type module, et peut donc être employé partout où ce dernier est attendu.

Un exemple est donné en 6.2.2 page suivante.

5.3.2 Composition parallèle

L'opérateur de composition parallèle est beaucoup plus souple, mais ne peut s'appliquer qu'aux signaux audio, car il modifie potentiellement le support temporel. Il effectue simplement la superposition d'un ensemble de signaux audio.

Sa sémantique est celle de *par* (cf. 3.2.5).

Sa syntaxe est simplement celle d'une liste :

```
sigs: une liste de signaux (audio)
```

Lorsqu'un module est attendu (altérations temporelles, paramètres de modules), une liste sera systématiquement interprétée comme une composition parallèle. Le résultat est du type module (audio).

Un exemple est donné en 6.3.3 page 38.

6 Exemple de référence

Nous allons maintenant illustrer le fonctionnement de FLSC au moyen d'un exemple qui correspond à un cas d'usage possible ; celui-ci est d'une simplicité minimale, afin de le rendre aussi accessible que possible. En référence à l'algorithme II.1 page suivante, nous en expliquerons l'essentiel, et donnerons son interprétation mathmématique.

Cet algorithme produit une séquence de sept hauteurs aléatoires, à des intervalles aléatoires. Chaque objet de cette séquence est une onde en dents-de-scie, amplifiée suivant une enveloppe ADSR.

Il est divisé en quatre parties :

1. définition des constantes employées
2. définition d'un instrument virtuel, à partir d'une enveloppe dynamique et d'un générateur de signal
3. définition des liste de paramètres sur lesquelles cet instrument sera itéré
4. finalement, application de la fonction instrument sur les listes de paramètres, ce qui produit un fragment audionumérique

6.1 Instrument virtuel

Nous allons nous focaliser sur la partie centrale de l'algorithme, intitulée 'composantes de l'instrument et définition', qui contient les éléments FLSC spécifiques nécessaires à notre objectif.

Cette partie est subdivisée en trois définitions :

- définition de l'envloppe dynamique `dyn`
- définition du générateur de signal `spect`
- assemblage de ces éléments en un instrument virtuel `instr`

Nous procèderons depuis les expressions intérieures vers l'extérieur, dans le but d'expliquer les effets des opérations successives sur les composantes élémentaires.

6.2 Enveloppe dynamique

6.2.1 Segments d'enveloppe

Les opérateurs `mseg` (cf. 5.1.3 page 32) et `mc` produisent respectivement un segment linéaire entre une valeur initiale et une valeur finale, et une valeur constante. Ces signaux sont associés aux objets $seg^O(a, b)$ et à un objet $const^O$ tel que :
$const^O(x)^R : t \mapsto x,\ x \in \mathbb{R}$

6.2.2 Composition séquentielle

Ces segments sont ensuite composés séquentiellement par l'opérateur `seq`
(cf. 5.3.1 page précédente), par l'expression :

```
(seq [(mseg 0 (* 2 amp)) (mseg (* 2 amp) amp)
      (mc amp) (mseg amp 0)])
```
{E1}

Cette expression produit un objet défini selon 3.2.4 comme :

$$\begin{cases} \Big(F : \big((o_0, \ldots, o_3), (T_0, T_4, (T_1, T_2, T_3))\big) \mapsto \\ \Big(\forall i \in \{0, \ldots, 3\} : t \in [T_i^R, T_{i+1}^R] \mapsto map_T\big(\tau \mapsto (i, \tau)\big)(o_i)^R(t)\Big), \\ P = (seg^O(0, 2 \cdot amp), seg^O(2 \cdot amp, amp), const^O(amp), seg^O(amp, 0)), \\ T = \Big(t \mapsto 0, t \mapsto t, \big((1, t \mapsto 0), (2, t \mapsto 0), (3, t \mapsto 0)\big)\Big)\Big) \end{cases}$$

Ceci signifie que les segments successifs se voient attribuer des indices successifs de signature temporelle abstraite, et que leur support est projeté dans l'intervalle abstrait du segment de signature associé.

Algorithme II.1 Cas d'usage simple

```
;;; 1) valeurs numérique
(define (end 2) ; durée des objets
        (amp 0.05) ; amplitude à l'entretien
        (bfreq 220) ; fréquence de base
        (numcomp 16) ; nombre de composantes
        ;; intervalle temporel aléatoire entre les objets
        (randstep (lambda () (++ (random 1.0))))
        (nboct 1.5) ; intervalle des hauteurs relatives alétoires des objets
        (numobj 7)) ; nombre d'objets

;;; 2) composantes de l'instrument et définition
(define
  ;; composante morphologique: enveloppe dynamique
  (dyn (sign [(* dur 0.05) (* dur 0.1) (* dur 0.75)] ; signature temporelle
         ;; composition séquentielle de segments
         (seq [(mseg 0 (* 2 amp)) (mseg (* 2 amp) amp)
               (mc amp) (mseg amp 0)])))
  ;; composante spectrale: composition parallèle de sinusoïdes
  (spect (lambda (tone) ; depend d'une hauteur relative aléatoire
           (map (lambda (i) ; itération sur une liste d'indices successifs
                  (m* (// i) ; l'amplitude de la i-ème composante est 1/i
                    ;; oscillateur de frequence i*bfreq*(2^tone)
                    (mosc (* i bfreq (** 2 tone)))))
                (range1 numcomp)))) ; indices {1..numcomp}
  ;; instrument virtuel
  (instr (lambda (date tone) ; depend de la date et de la hauteur relative
           (delay date ; situation temporelle
             (base [0 end] ; durée
               ;; produit de la dynamique et du spectre
               (m* dyn (spect tone)))))))

;;; 3) paramètres de l'instrument
(define
  ;; liste aléatoire de dates: sommation de valeurs aléatoires
  (dates (reduce (lambda (acc val) (append acc (+ (last acc) val)))
           (fill randstep (-- numobj)) [0]))
  ;; liste aléatoire de hauteurs relatives
  (tones (fill (lambda () (random nboct)) numobj)))

;;; 4) resultat: itération de l'instrument sur les dates et hauteurs
(map instr dates tones)
```

6.2.3 Application de signature

L'enveloppe dynamique `dyn` est ensuite obtenue par l'application d'une signature abstraite (cf. 5.2 page 33) sur l'expression précédente `{E1}` :

```
(dyn (sign [(* dur 0.05) (* dur 0.1) (* dur 0.75)] {E1}))          {E2}
```

La sous-expression `(* dur x)` est équivalente à $(t \mapsto t) \cdot x = (t \mapsto t \cdot x)$ (cf. 3.1.1 page 26).
En conséquence, l'objet temporel associé est défini selon 3.2.3 page 28 comme :
Soit $(T_n)_4 = (t \mapsto 0, t \mapsto t \cdot 0.05, t \mapsto t \cdot 0.1, t \mapsto t \cdot 0.75, t \mapsto t)$ *dans* :

$$map_T \left(\begin{cases} \tau & \mapsto \tau \\ (n, \tau) & \mapsto \tau(T_{n+1} - T_n) + T_n \end{cases} \right)$$

$$\left(\begin{cases} \Big(F : \big((o_0, \ldots, o_3), (T_0, T_4, (T_1, T_2, T_3))\big) \mapsto \\ \Big(\forall i \in \{0, \ldots, 3\} : t \in [T_i^R, T_{i+1}^R] \mapsto map_T\big(\tau \mapsto (i, \tau)\big)(o_i)^R(t)\Big), \\ P = (seg^O(0, 2 \cdot amp), seg^O(2 \cdot amp, amp), const^O(amp), seg^O(amp, 0)), \\ T = \Big(t \mapsto 0, t \mapsto t, \big((1, t \mapsto 0), (2, t \mapsto 0), (3, t \mapsto 0)\big)\Big) \end{cases} \right)$$

Cette expression peut être ramenée à :

$$\begin{cases} \Big(F : \big((o_0, \ldots, o_3), (T_0, T_4, (T_1, T_2, T_3))\big) \mapsto \\ \Big(\forall i \in \{0, \ldots, 3\} : t \in [T_i^R, T_{i+1}^R] \mapsto map_T\big(\tau \mapsto \tau(T_{i+1} - T_i) + T_i\big)(o_i)^R(t)\Big), \\ P = \big(seg^O(0, 2 \cdot amp), seg^O(2 \cdot amp, amp), const^O(amp), seg^O(amp, 0)\big), \\ T = \Big(t \mapsto 0, t \mapsto t, \big(t \mapsto t \cdot 0.05, t \mapsto t \cdot 0.1, t \mapsto t \cdot 0.75\big)\Big)\Big) \end{cases}$$

Ceci signifie que les segments sont projetés dans des supports successifs, définis relativement au support abstrait englobant.

6.2.4 Objet temporel résultant

Etant donnée l'équation donnée en 5.1.3 page 32 :

$$seg^O(a, b) = \begin{cases} \Big(\Big(F : \big((a, b), (T_0, T_f)\big) \mapsto \\ \big(t \in [T_0^R, T_f^R] \mapsto a + \frac{t - T_0^R}{T_f^R - T_0^R} \cdot (b - a)\big)\Big), \quad ; \\ P = (a, b), T = (t \mapsto 0, t \mapsto t)\Big) \end{cases}$$

et le fait que $const^O(x)$ peut être considéré comme équivalent à $seg^O(x, x)$, l'expression ci-dessus peut être reformulée comme :

$$\begin{cases} \Big(F : \big((x_0, \ldots, x_4), (T_0, T_4, (T_1, T_2, T_3))\big) \mapsto \\ \Big(\forall i \in \{0, \ldots, 3\} : t \in [T_i^R, T_{i+1}^R] \mapsto x_i + \frac{t - T_i^R}{T_{i+1}^R - T_i^R} \cdot (x_{i+1} - x_i)\Big), \\ P = (0, 2 \cdot amp, amp, amp, 0), \\ T = \Big(t \mapsto 0, t \mapsto t, \big(t \mapsto t \cdot 0.05, t \mapsto t \cdot 0.1, t \mapsto t \cdot 0.75\big)\Big)\Big) \end{cases}$$

6.3 Générateur de signal

6.3.1 Oscillateur élémentaire

L'élément spectral de base est produit par l'expression :

```
(mosc (* i bfreq (** 2 tone)))          {E3}
```

Ceci correspond à un objet oscillateur sinusoïdal, défini en 4.3.4 page 31 comme :

$$mod^O_{sin}(const^O(i \cdot bfreq \cdot 2^{tone}), 0) = \begin{cases} \Big(F : \big((f,\phi),(T_0,T_f)\big) \mapsto \\ \Big(t \in [T_0^R, T_f^R] \mapsto sin\big(\int_{T_0}^t f^R(\tau)\cdot d\tau + \phi\big)\Big), \\ P = \big(const^O(i \cdot bfreq \cdot 2^{tone}), 0\big), T = (t \mapsto 0, t \mapsto t)\Big) \end{cases}$$

La fréquence de base `bfreq` est définie dans la partie 1, l'indice de composante `i` est la variable d'itération de la fonction `map` englobante, et `tone` est le paramètre de la fonction englobante.

6.3.2 Amplitude des composantes

Les composantes reçoivent une amplitude par dérivation de l'expression précédente `{E3}` vers :

```
(m* (// i) {E3})                                                    {E4}
```

Ceci signifie que le signal produit par chaque composante est multiplié par $\frac{1}{i}$, ce qui résulte en l'objet suivant :

$$\begin{cases} \Big(F : \big((a,b),(T_0,T_f)\big) \mapsto \\ \Big(t \in [T_0^R, T_f^R] \mapsto a^R(t)\cdot b^R(t)\Big), \\ P = \left(const^O(\frac{1}{i}), \begin{cases} \Big(F : \big((f,\phi),(T_0,T_f)\big) \mapsto \\ \Big(t \in [T_0^R, T_f^R] \mapsto sin\big(\int_{T_0}^t f^R(\tau)\cdot d\tau + \phi\big)\Big), \\ P = \big(const^O(i \cdot bfreq \cdot 2^{tone}), 0\big), T = (t \mapsto 0, t \mapsto t)\Big) \end{cases}\right), T = (t \mapsto 0, t \mapsto t)\Big) \end{cases}$$

Ceci peut être ramené à :

$$\begin{cases} \Big(F : \big((f,\phi),(T_0,T_f)\big) \mapsto \\ \Big(t \in [T_0^R, T_f^R] \mapsto \frac{1}{i}\cdot sin\big(\int_{T_0}^t f^R(\tau)\cdot d\tau + \phi\big)\Big), \\ P = \big(const^O(i \cdot bfreq \cdot 2^{tone}), 0\big), T = (t \mapsto 0, t \mapsto t)\Big) \end{cases}$$

6.3.3 Itération

Les composantes sont produite par itération sur l'expression `{E4}`, comme suit :

```
(map (lambda (i) {E4}) (range1 numcomp))                            {E5}
```

Ceci produit une liste de signaux, avec `i` entre `1` et `numcomp` (défini en partie 1), qui sera plus tard interprété comme une compostion parallèle (cf. 5.3.2 page 34). L'objet correspondant à cette composition est défini suivant 3.2.5 comme :

$$Soit\ o_i = \begin{cases} \Big(F : \big((f,\phi),(T_0,T_f)\big) \mapsto \\ \Big(t \in [T_0^R, T_f^R] \mapsto \frac{1}{i}\cdot sin\big(\int_{T_0}^t f^R(\tau)\cdot d\tau + \phi\big)\Big), \\ P = \big(const^O(i \cdot bfreq \cdot 2^{tone}), 0\big), T = (t \mapsto 0, t \mapsto t)\Big) \end{cases}$$

$$dans : \begin{cases} \Big(F : \big((o_1,\ldots,o_{numcomp}),(T_0,T_f)\big) \mapsto \\ \big(\forall t \in [T_0,T_f] : t \mapsto \sum_{i=1}^{numcomp} ext|_{T_f}^{T_0} o_i^R(t)\big), \\ P = (o_1,\ldots,o_{numcomp}), T = (t \mapsto 0, t \mapsto t)\Big) \end{cases}$$

Cette expression peut être considérée équivalente à :

$$\begin{cases} \Big(F : \big((f,\phi),(T_0,T_f)\big) \mapsto \\ \Big(\forall t \in [T_0,T_f] : t \mapsto \sum_{i=1}^{numcomp} \frac{1}{i}\cdot sin\big(\int_{T_0}^t i\cdot f^R(\tau)\cdot d\tau + \phi\big)\Big), \\ P = \big(const^O(bfreq \cdot 2^{tone}), 0\big), T = (t \mapsto 0, t \mapsto t)\Big) \end{cases}$$

Des objets de ce type sont produits par la fonction `spect`, qui dépend de `tone`, et qui est exprimée comme suit :

```
(spect (lambda (tone) {E5}))
```
{E6}

6.4 Assemblage des composantes

6.4.1 Application de la dynamique au générateur de signal

L'objet temporel visé peut être exprimé par l'expression suivante :

```
(m* dyn (spect tone))
```
{E7}

La variable `tone` est un paramètre de la fonction englobante.
Cet objet est défini comme suit :

$$\left\{\begin{array}{l} \Big(F:\big((a,b),(T_0,T_f)\big)\mapsto\Big(t\in[T_0^R,T_f^R]\mapsto a^R(t)\cdot b^R(t)\Big), \\ P=\left(\left\{\begin{array}{l} \Big(F:\Big((x_0,\ldots,x_4),(T_0,T_4,(T_1,T_2,T_3))\Big)\mapsto \\ \Big(\forall i\in\{0,\ldots,3\}:t\in[T_i^R,T_{i+1}^R]\mapsto x_i+\frac{t-T_i^R}{T_{i+1}^R-T_i^R}\cdot(x_{i+1}-x_i)\Big), \\ P=(0,2\cdot amp,amp,amp,0), \\ T=\Big(t\mapsto 0,t\mapsto t,\big(t\mapsto t\cdot 0.05,t\mapsto t\cdot 0.1,t\mapsto t\cdot 0.75\big)\Big) \end{array}\right., \right. \\ \left.\left\{\begin{array}{l} \Big(F:\big((f,\phi),(T_0,T_f)\big)\mapsto \\ \Big(\forall t\in[T_0,T_f]:t\mapsto \sum_{i=1}^{numcomp}\frac{1}{i}\cdot sin\big(\int_{T_0}^t i\cdot f^R(\tau)\cdot d\tau+\phi\big)\Big), \\ P=\big(const^O(bfreq\cdot 2^{tone}),0\big),T=(t\mapsto 0,t\mapsto t)\Big) \end{array}\right)\right. \\ \left.,T=(t\mapsto 0,t\mapsto t)\right)\Big) \end{array}\right.$$

Il peut être simplifié sous la forme suivante :

$$\left\{\begin{array}{l} \Big(F:\Big((f,\phi,x_0,\ldots,x_4),(T_0,T_4,(T_1,T_2,T_3))\Big)\mapsto \\ \Big(\forall i\in\{0,\ldots,3\}:t\in[T_i^R,T_{i+1}^R]\mapsto \big(x_i+\frac{t-T_i^R}{T_{i+1}^R-T_i^R}\cdot(x_{i+1}-x_i)\big)\cdot \\ \sum_{i=1}^{numcomp}\frac{1}{i}\cdot sin\big(\int_{T_0}^t i\cdot f^R(\tau)\cdot d\tau+\phi\big)\Big), \\ P=\big(const^O(bfreq\cdot 2^{tone}),0,0,2\cdot amp,amp,amp,0\big), \\ T=\Big(t\mapsto 0,t\mapsto t,\big(t\mapsto t\cdot 0.05,t\mapsto t\cdot 0.1,t\mapsto t\cdot 0.75\big)\Big)\Big) \end{array}\right.$$

6.4.2 Projection dans un support temporel

On attribue alors un support temporel à l'objet (cf. 5.2 page 33), par l'expression :

```
(base [0 end] {E7})
```
{E8}

Celle-ci est définie en 3.2.2 page 28 comme :

$$map_T\Big(\tau\mapsto\tau(end-0)+0\Big)\left(\left\{\begin{array}{l} \Big(F:\Big((f,\phi,x_0,\ldots,x_4),(T_0,T_4,(T_1,T_2,T_3))\Big)\mapsto \\ \Big(\forall i\in\{0,\ldots,3\}:t\in[T_i^R,T_{i+1}^R]\mapsto \big(x_i+\frac{t-T_i^R}{T_{i+1}^R-T_i^R}\cdot(x_{i+1}-x_i)\big)\cdot \\ \sum_{i=1}^{numcomp}\frac{1}{i}\cdot sin\big(\int_{T_0}^t i\cdot f^R(\tau)\cdot d\tau+\phi\big)\Big), \\ P=\big(const^O(bfreq\cdot 2^{tone}),0,0,2\cdot amp,amp,amp,0\big), \\ T=\Big(t\mapsto 0,t\mapsto t,\big(t\mapsto t\cdot 0.05,t\mapsto t\cdot 0.1,t\mapsto t\cdot 0.75\big)\Big)\Big) \end{array}\right)$$

Etant donné que `end` est défini comme `2` dans la partie 1, ceci revient à :

$$\left\{\begin{array}{l}\Big(F:\Big((f,\phi,x_0,\ldots,x_4),(T_0,T_4,(T_1,T_2,T_3))\Big)\mapsto\\ \Big(\forall i\in\{0,\ldots,3\}:t\in[T_i^R,T_{i+1}^R]\mapsto\big(x_i+\frac{t-T_i^R}{T_{i+1}^R-T_i^R}\cdot(x_{i+1}-x_i)\big)\cdot\\ \sum_{i=1}^{numcomp}\frac{1}{i}\cdot sin\big(\int_{T_0}^t i\cdot f^R(\tau)\cdot d\tau+\phi\big)\Big),\\ P=\big(const^O(bfreq\cdot 2^{tone}),0,0,2\cdot amp,amp,amp,0\big),\\ T=\Big(t\mapsto 0,t\mapsto 2,\big(t\mapsto 0.1,t\mapsto 0.2,t\mapsto 1.5\big)\Big)\Big)\end{array}\right.$$

Les valeurs temporelles qui définissent cette objet sont constantes, et il est donc réalisable. De plus, elles sont ordonnées de façon croissante, et ainsi l'objet est temporellement cohérent. En conséquence, il s'agit d'un fragment.

6.4.3 Situation temporelle

Le fragment peut être situé à une date précise (donnée par le paramètre `date`) dans la temporalité globale (cf. 5.2 page 33) :

```
(delay date {E8})                                                    {E9}
```

Ceci est défini en 3.2.6 page 29 comme :

$$map_T\Big(\tau\mapsto\tau+date\Big)\left(\left\{\begin{array}{l}\Big(F:\Big((f,\phi,x_0,\ldots,x_4),(T_0,T_4,(T_1,T_2,T_3))\Big)\mapsto\\ \Big(\forall i\in\{0,\ldots,3\}:t\in[T_i^R,T_{i+1}^R]\mapsto\big(x_i+\frac{t-T_i^R}{T_{i+1}^R-T_i^R}\cdot(x_{i+1}-x_i)\big)\cdot\\ \sum_{i=1}^{numcomp}\frac{1}{i}\cdot sin\big(\int_{T_0}^t i\cdot f^R(\tau)\cdot d\tau+\phi\big)\Big),\\ P=\big(const^O(bfreq\cdot 2^{tone}),0,0,2\cdot amp,amp,amp,0\big),\\ T=\Big(t\mapsto 0,t\mapsto 2,\big(t\mapsto 0.1,t\mapsto 0.2,t\mapsto 1.5\big)\Big)\Big)\end{array}\right.\right)$$

Toutes les valeurs temporelles étant constantes, cette expression peut être ramenée à :

$$\left\{\begin{array}{l}\Big(F:\Big((f,\phi,x_0,\ldots,x_4),(T_0,T_4,(T_1,T_2,T_3))\Big)\mapsto\\ \Big(\forall i\in\{0,\ldots,3\}:t\in[T_i^R,T_{i+1}^R]\mapsto\big(x_i+\frac{t-T_i^R}{T_{i+1}^R-T_i^R}\cdot(x_{i+1}-x_i)\big)\cdot\\ \sum_{i=1}^{numcomp}\frac{1}{i}\cdot sin\big(\int_{T_0}^t i\cdot f^R(\tau)\cdot d\tau+\phi\big)\Big),\\ P=\big(const^O(bfreq\cdot 2^{tone}),0,0,2\cdot amp,amp,amp,0\big),\\ T=\Big(t\mapsto date,t\mapsto date+2,\big(t\mapsto date+0.1,t\mapsto date+0.2,t\mapsto date+1.5\big)\Big)\Big)\end{array}\right.$$

6.4.4 Instrument résultant

L'instrument virtuel `instr` peut alors être exprimé comme une fonction qui dépend de `date` et `tone` :

```
(instr (lambda (date tone) {E10}))                                   {E10}
```

L'expression globale est la suivante :

$$instr(date,tone)=\left\{\begin{array}{l}\Big(F:\Big((f,\phi,x_0,\ldots,x_4),(T_0,T_4,(T_1,T_2,T_3))\Big)\mapsto\\ \Big(\forall i\in\{0,\ldots,3\}:t\in[T_i^R,T_{i+1}^R]\mapsto\big(x_i+\frac{t-T_i^R}{T_{i+1}^R-T_i^R}\cdot(x_{i+1}-x_i)\big)\cdot\\ \sum_{i=1}^{numcomp}\frac{1}{i}\cdot sin\big(\int_{T_0}^t i\cdot f^R(\tau)\cdot d\tau+\phi\big)\Big),\\ P=\big(const^O(bfreq\cdot 2^{tone}),0,0,2\cdot amp,amp,amp,0\big),\\ T=\Big(t\mapsto date,t\mapsto date+2,\big(t\mapsto date+0.1,t\mapsto date+0.2,t\mapsto date+1.5\big)\Big)\Big)\end{array}\right.$$

Cet objet peut être réalisé, ce qui produit l'expression finale :

$$instr(date, tone)^R = \begin{cases} Let & \begin{cases} (T_n)_5 = & (date, date + 0.1, date + 0.2, date + 1.5, date + 2) \\ (x_n)_5 = & (0, 2 \cdot amp, amp, amp, 0) \end{cases} \\ in: & \begin{cases} \Big(\forall i \in \{0, \ldots, 3\} : t \in [T_i^R, T_{i+1}^R] \mapsto \big(x_i + \frac{t - T_i^R}{T_{i+1}^R - T_i^R} \cdot (x_{i+1} - x_i)\big) \cdot \\ \sum_{i=1}^{numcomp} \frac{1}{i} \cdot sin\big(i \cdot bfreq \cdot 2^{tone} \cdot (t - T_0)\big)\Big) \end{cases} \end{cases}$$

Ceci est en effet l'expression d'un oscillateur en dents-de-scie, avec une dynamique ADSR.

7 Conclusion

Nous avons exposé ici la structure et les axiomes d'un langage informatique permettant de créer et manipuler les éléments fondamentaux d'un algortihme de synthèse sonore ; notamment, nous avons montré comment la notion de temporalité peut être exprimée par des opérations élémentaires, indépendamment des objets sur lesquels elle opère. L'équivalence des expressions de ce langage avec la formulation mathématique d'un signal électroacoustique envoyé à un amplificateur peut être déterminée au moyen des règles que nous avons exposées ; la cohérence de ces expressions dépend essentiellement du respect des contraintes de typage, ainsi que des contraintes pesant sur les opérations temporelles.

Ce langage nous permettra par la suite d'exprimer, d'après les théories musicologiques existantes, les schémas sous-jacents à la production d'objets sonores de synthèse, dont nous souhaitons étudier la compréhension par la perception humaine. Parmi les langages existants pour la synthèse sonore, il y en a peu dont les productions sont pleinement spécifiées par un formalisme mathématique sans ambigüité ; nous espérons donc pouvoir généraliser l'expérience acquise dans de nombreux autres langages, en en établissant l'équivalence.

Chapitre III

Implémentation (SuperCollider)

Le langage formel donné ci-dessus ne resterait que lettre morte, si il n'était pas possible de réaliser effectivement les structures de synthèse sonore qu'il décrit. En conséquence, nous avons produit une implémentation de ce langage, qui agit comme une preuve *par le fait* de sa validité pratique.

Nous allons donc montrer ici par quels moyens, en pratique, une expression en algèbre fonctionnelle FLSC peut être interprétée comme un enregistrement audionumérique.

Nous utiliserons pour cela le langage SuperCollider (abrégé ci-après SC) [26], et des outils standards tels que `flex` et `bison` [13].

Le code source de l'application résultante est disponible sous la forme d'un dépôt Git hébergé par GitHub [18].

Globalement la stratégie consiste à traduire l'expression, au moyen des outils habituels, sous la forme d'un *arbre sémantique*, donnée comme un arbre de constructeurs de classes SC dédiées. Les éléments de cet arbre seront dotés de méthodes permettant son interprétation récursive comme un enregistrement audionumérique (mais également, potentiellement, comme plusieurs types de représentations graphiques). L'arbre sémantique peut ensuite être importé dans SC puis évalué pour produire l'objet audionumérique spécifié.

1 Introduction à SuperCollider

Afin que le processus de traduction de l'expression fonctionnelle sous la forme d'un objet manipulable par SC soit plus facile à saisir, nous allons tout d'abord commencer par présenter les concepts fondamentaux de la programmation en SC.

Ce langage s'articule autour de deux éléments logiciels :

le serveur de synthèse sonore dont il existe deux versions, `scsynth` (la version originelle) et `supernova` (qui a été réécrit pour exploiter les architectures parallèles). Du point de vue de l'utilisateur, ces deux programmes fonctionnent de façon analogue : ils permettent d'effectuer de façon optimisée des calculs de synthèse sonore, et sont pilotés au moyen de messages OSC [23] qui correspondent aux différentes opérations supportées.

l'interpréteur de commandes `sclang`, qui permet de dialoguer avec le serveur au moyen d'un langage orienté objet réactif et temporalisé, et ainsi d'écrire des programmes qui peuvent être relativement élaborés sans pour autant nécessiter de compilation (les processus de synthèse sonore n'étant pas exécutés dans l'interpréteur).

1.1 Le serveur

Essentiellement, le serveur permet de créer, démarrer, contrôler, arrêter et détruire des modules de synthèse ; ceux-ci sont spécifiés par un bytecode qui correspond à des objets partagés précompilés. Les modules de synthèse peuvent communiquer par des variables de liaison (traditionnellement nommés *bus*, par référence aux tables de mixage).

Les spécifications de modules de synthèse peuvent être téléversés sur le serveur à partir de l'interpréteur ; ces spécifications sont nécessairement algorithmiquement invariantes (mais peuvent admettre des paramètres scalaires). Les bus étant représentés côté serveur par un simple indice, il est possible de passer un bus en paramètre à une instance de module de synthèse.

Les éléments du serveur sont représentés dans l'interpréteur par des abstractions qui permettent de les manipuler sans connaître le langage interne du serveur. En conséquence, nous ne rentrerons pas dans la description du dialecte OSC employé pour communiquer avec le serveur.

1.2 L'interpréteur

L'utilisateur travaille dans la quasi-totalité des cas uniquement au moyen de l'interpréteur. Celui-ci emploie un langage orienté objet à typage dynamique, avec de nombreuses classes de bibliothèque.

Les classes SC commencent toujours par une majuscule (comme par exemple `Score` qu'on verra plus loin), ce qui permettra de les distinguer dans la suite des éléments du langage fonctionnel qui sont en minuscules.

Les méthodes et les champs des objets sont appelées au moyen d'un opérateur noté par un point ; nous noterons les méthodes et champs d'une classe par le nom de la classe, suivi d'un point et du nom de la méthode (`Score.play` par exemple), bien qu'en ce qui concerne les méthodes et champs d'instance on utilise en pratique l'identificateur de l'instance et non le nom de classe (`myScore.play` par exemple).

Lorsque le nom de la méthode est absent, il s'agit implicitement du constructeur (qui s'appelle habituellement `new` : `Score(...)` est équivalent à `Score.new(...)`).

Les noms de variable sont introduits par le mot-clef `var`, et commencent par une minuscule ; elles ne sont accessibles que dans le bloc dans lequel elles ont été définies. Néanmoins les variables dont le nom est composé d'une seule lettre sont des variables globales, c'est à dire qu'elles sont accessibles n'importe où. En particulier la variable `s` désigne `Server.default`, le serveur par défaut (voir 1.3.8).

1.3 Classes employées

Nous allons ici employer un certain nombre de classes d'extension que nous avons définies en adéquation avec le contexte. Ces classes seront nommés `FLSC_<nom de classe>`, pour *Functional Language for Sound Composition*, ce qui permettra de les distinguer des classes de la bibliothèque SC. Les extensions seront détaillées au cours de l'exposé du processus de traduction.

Nous utilisons également des classes de bibliothèque. Les principales sont présentées ici succintement (se référer à la documentation SC pour plus de détails).

1.3.1 Nombres, booléens et symboles

Les constantes sont des nombres, des booléens et des symboles.

La classe `Number` décrit les nombres, nous ne nous intéresserons ici qu'à sa sous-classe `SimpleNumber` qui décrit les nombres mono-dimensionnels (les autres étant `Polar` et `Complex`).

Une constante numérique s'écrit (en expression régulière) :

`-?[0-9]+(\.[0-9]+)?(e-?[0-9]+)?(pi)?`

Ce qui signifie qu'un nombre est composé :

1. d'un signe moins optionnel
2. d'une valeur entière
3. d'une valeur décimale optionnelle
4. d'un exposant optionnel
5. d'une multiplication par π optionnelle

Les nombres ont des méthodes pour l'arithmétique, la conversion vers le sous-type entier (`Integer`), etc...

Les booléens, du type `Boolean`, n'ont que deux réalisations : `true` et `false`. Ils sont utilisés notamment dans les expressions conditionnelles, et donc dans les opérateurs relationnels qui les alimentent.

Les symboles, de type `Symbol`, sont uniques (c'est à dire que deux symboles identiques sont le même objet), et sont notés selon les expression régulières `'.+'` ou `\\.+`.

1.3.2 Collections : Array, Dictionnary, Set

Les tableaux en SC sont de la classe `Array`. Cette classe est souvent notée par des raccourcis syntaxiques : `[elt, ...]` est un `Array` et `Array.at(i)` qui retourne l'élément d'indice `i` est noté `Array[i]`. `Array.newClear(n)` permet de créer un tableau à `n` éléments.

Les `Array` peuvent être modifiés, notamment par la méthode `Array.add` qui ajoute un élément en queue et `Array ++ Array`, la concaténation.

Des méthodes d'itération comme `Array.do` ou `Array.collect` sont disponibles. La première évalue pour chaque élément une fonction (avec cet élément pour argument, ainsi que son indice) ; la deuxième est analogue, mais retourne un tableau contenant les résultats successifs de la fonction.

La méthode `Array.inject` applique récursivement une fonction sur une valeur accumulée et les éléments de cette collection. Ainsi `Array.inject(startVal, {|a, b| body})` applique `body` récursivement sur tous les éléments `b` du tableau, à partir de la valeur `a` accumulée, intialement égale à `startVal`.

La classe `Dictionnary` représente un dictionnaire associatif et fonctionne de façon analogue, mis à part que les clefs sont des objets arbitraires (et non des entiers successifs). La classe `IdentityDictionnary` est une variante particulière qui compare ses clefs sur le critère que les objets sont les mêmes (et non qu'ils on la même valeur).

La classe `Set` définit un ensemble (ce qui permet d'éviter les doublons). On peut ajouter ou supprimer des éléments, et itérer sur ces collections, comme sur n'importe quelle autre.

1.3.3 Function

Les fonctions en SC sont également des objets (de classe `Function`). Une fonction est notée `{|arg ...| body}` et possède une méthode `Function.value(arg, ...)` qui permet de l'appeler. En raison du typage dynamique, cela permet un polymorphisme entre les fonctions et les autres objets puisque la superclasse `Object` définit la méthode `Object.value` qui renvoie (à défaut de surcharge) l'objet lui-même.

1.3.4 Synth et SynthDef

La classe `Synth` décrit une instance de module de synthèse et la classe `SynthDef` la spécification d'un type de module.

`SynthDef` possède deux méthodes qui nous intéressent, `SynthDef.new` (son constructeur) et `SynthDef.add` (qui permet de téléverser la spécification sur le serveur).

`SynthDef.new` prend en argument le nom de la spécification (sous la forme d'un `Symbol`, un symbole), ainsi qu'une fonction `uGenGraphFunc` qui retourne un graphe de calcul (un graphe de `UGen`, qui sont les éléments de base de la synthèse sonore en SC). Les arguments de cette fonction sont convertis en entrées dynamiques (`Control`), qui permettent d'interagir avec les instances.

Les instances sont crées par `Synth.new`, qui prend en argument le nom de la spécification ainsi qu'un tableau de paires `[arg, val, ...]` afin d'initialiser les entrées. Cet appel instancie le module et le démarre immédiatement. Les entrées peuvent être réaffectées plus tard au moyen de `Synth.set(arg, val, ...)`. Enfin, le module peut être arrêté et détruit par `Synth.free`.

1.3.5 UGen et Control

Les `UGen` (pour *Unit Generator*) sont les éléments de base de la synthèse sonore en SC. Ils correspondent aux primitives de génération de signal, ainsi qu'aux opérations algébriques sur celles-ci (sous-type `BinaryOpUGen`). Ils apparaissent essentiellement dans le corps de la `uGenGraphFunc` qui définit une `SynthDef`, et constituent le graphe de calcul associé à celle-ci.

Les `UGen` fonctionnent à différentes fréquences d'échantillonage, notamment les fréquences audio, de contrôle, ou statiques. Le plus souvent les constructeurs d'un type donné sont `UGen.ar` (*audio rate*), `UGen.kr` (*control rate*), ou `UGen.ir` (*initialisation rate*), pour ces fréquences respectives. La méthode `UGen.rate` permet de déterminer la fréquence d'échantillonnage des signaux, en retournant respectivement `'audio'`, `'control'` ou `'scalar'`.

Les nombres ont également cette méthode mesurant le taux d'échantillonnage : `SimpleNumber.rate`, qui retourne `'scalar'`, et qui permet de les distinguer des types retournant des valeurs dynamiques (variantes au cours du temps).

Les `Control` sont un type particulier de `UGen`, qui correspond aux paramètres des `Synth` associés à la `SynthDef`. En effet les valeurs passées en argument aux `Synth` peuvent être dynamiques, et être mises à jour par des messages `Synth.set(arg, val, ...)`. Les `Control` sont liés à un `Symbol` qui est utilisé pour les désigner. Dans la `uGenGraphFunc`, les arguments sont convertis en `Control`.

`Symbol.(ar|kr|ir)` permet d'obtenir un `Control` au taux d'échantillonnage souhaité, avec pour nom le `Symbol` en question. Cette instruction n'est valide qu'à l'intérieur de la `uGenGraphFunc` d'une `SynthDef`, et permet d'ajouter des contrôles dynamiques à des modules, au besoin.

1.3.6 Score

Un ensemble de messages OSC avec leurs dates d'envoi peut être représenté par un objet `Score` (partition virtuelle). Les messages sont alors regroupés en *paquets* (qui sont composés d'une date et d'une liste de messages). Cette classe permet de traiter à un haut niveau l'organisation d'un ensemble de messages, et sera parfaitement adaptée dans le cas qui nous occupe.

Outre son constructeur (qui retourne une partition à partir d'une liste de paquets), elle dispose des méthodes `Score.add` (ajout d'un paquet), `Score.sort` (tri chronologique des paquets), `Score.play` (exécution de la partition en temps réel) et `Score.recordNRT` (rendu dans un enregistrement hors temps réel).

1.3.7 Routine

L'interpréteur peut créer des fils d'exécution (*threads*) séparés, qui peuvent alors se dérouler de façon synchrone. Ils sont représentés par des objets du type `Routine`, qui sont définis par une `Function`.

A l'intérieur de cette fonction, un certain nombre de choses sont possibles : par exemple l'appel à `SimpleNumber.wait` qui attend pendant un certain nombre d'unités de temps (qui dépendent du `TempoClock` sur lequel est calée la `Routine`). Ainsi, il est possible d'effectuer une programmation synchrone de façon à effectuer un séquencement des opérations.

Les `Routine` sont lancées par `Routine.play` ; à partir de là elles se poursuivent dans un fil d'exécution séparé et le programme principal continue de façon asynchrone. Cela permet la plupart du temps d'éviter un blocage du fil d'exécution principal.

1.3.8 Server

Le serveur lui-même est représenté par l'abstraction `Server`, qui possède des méthodes pour démarrer et arrêter (`Server.boot` et `Server.quit`).

Il y a également des méthodes synchrones qui servent à attendre l'exécution des instructions par le processus système correspondant (`Server.bootSync` pour le démarrage et `Server.sync` dans les autres cas, notamment le chargement de `SynthDef`). Ces méthodes ne sont valides qu'à l'intérieur d'une `Routine`.

Des listes de messages OSC, pouvant être envoyées au serveur pour le contrôler, peuvent être obtenues à partir des appels standard (`Synth.new`, etc...) grâce à la méthode `Server.makeBundle`. Après ajout d'une date, ces listes peuvent s'intégrer dans un `Score`, nous ferons donc abstraction du dialecte OSC sous-jacent.

1.4 Exemple de code SuperCollider

L'algorithme ci-dessous montre une réalisation élémentaire en SuperCollider, avec création d'une `SynthDef` (oscillateur sinusoïdal avec enveloppe dynamique trapézoïdale) et synthèse d'un objet audionumérique au moyen d'un objet `Score`.

L'algorithme suit le schéma suivant :

1. Création d'une `SynthDef` (algorithme III.1)
2. Création d'un `Score` contenant les messages de début et de fin (algorithme III.2)
3. Exécution de la partition virtuelle (algorithme III.3) :
   (a) démarrage du serveur
   (b) chargement de la `SynthDef`
   (c) lecture du `Score`
   (d) fermeture du serveur

Algorithme III.1 Exemple SC : partie 1

```
// début du programme
(
// créer les variables nécessaires
var def, score, synth, start, end;

/*
def est notre SynthDef (oscillateur sinusoidal à enveloppe trapézoidale)
son nom est 'example' (un Symbol)
ses arguments sont:
- out (le Bus de sortie, 0 est la première sortie système)
- freq (la fréquence)
- amp (l'amplitude)
- dur (la durée jusqu'au début de la chute)
*/
def = SynthDef('example', {|out = 0, freq = 220, amp = 0.5, dur = 1|
        // Out.ar spécifie le Bus de sortie,
        // SinOsc.ar est un oscillateur sinusoidal
        Out.ar(out, SinOsc.ar(freq, 0, amp) *
                // On multiplie l'oscillateur par une enveloppe trapézoidale
                EnvGen.kr(Env.linen(0.01, dur - 0.01, 0.5, 1, 'lin')));
});
```

Ici on affecte à `def` une `SynthDef` composée du produit d'un oscillateur sinusoïdal `SinOsc.ar` avec une enveloppe dynamique `EnvGen.kr(Env.linen(...))`. Le tout est envoyé sur un `Bus` de sortie audio par `Out.kr`.

Algorithme III.2 Exemple SC : partie 2

```
// score est notre objet Score
score = Score();

// start est le message de démarrage du synthétiseur
// le paramètre false signifie: ne pas envoyer au serveur,
// simplement créer un message
start = s.makeBundle(false, {
        // on affecte synth à l'objet Synth correspondant
        synth = Synth('example', ['freq', 330]);
});

// end est le message de fin
// on fait appel à la variable synth affectée précédemment
end = s.makeBundle(false, {
        synth.free;
});

// on ajoute les deux messages dans score, avec les dates souhaitées
// un bundle est un Array de messages
// Array.++ est la concaténation
// un élément de Score est une date suivie d'un bundle
score.add([1] ++ start);
score.add([2.5] ++ end);

// on trie score chronologiquement
// (ce n'est pas nécessaire ici, mais toujours plus prudent)
score.sort;
```

Ici on affecte à `score` un `Score` vide, auquel on ajoute deux messages `start` et `end`, créés avec `Server.makeBundle` ; ils sont simplement des listes composées d'un seul message.
`Server.makeBundle` prend comme premier argument `false`, ce qui signifie que le message ne doit pas être envoyé au serveur, mais simplement créé.
Lorsqu'on appelle `Score.add`, on ajoute au début de la liste la date d'exécution avec l'opérateur de concaténation.
On termine en triant chronologiquement `score`, ce qui n'est pas nécessaire dans ce cas mais peut l'être lorsqu'on ne connaît pas l'ordre des messages ajoutés.

Algorithme III.3 Exemple SC : partie 3

```
// l'environnement Routine est nécessaire
// pour utiliser les fonctions de synchronisation
// il s'agit d'un fil d'exécution séparé et temporalisé
Routine({

        // on démarre le serveur et on attend la synchronisation
        s.bootSync;

        // on ajoute la SynthDef
        def.add;

        // on attend la synchronisation
        // (ajouter une SynthDef prend du temps)
        s.sync;

        // maintenant on peut lancer le Score
        score.play;

        // on attend la fin du Score pour sortir
        // on ajoute une seconde pour éviter un artefact
        (score.endtime + 1).wait;

        // fermer le serveur
        s.quit;

        // Routine.play exécute la Routine
}).play;

// fin du programme
)
```

Finalement on utilise une `Routine` pour faire appel aux fonctionnalités de synchronisation de SC.

1. On démarre le serveur avec `Server.bootSync`
2. On charge la `SynthDef` avec `SynthDef.add`, puis on attend la synchronisation avec `Server.sync`
3. On lance le `Score` avec `Score.play`
4. On attend la fin du `Score` avec `SimpleNumber.wait` avant de fermer le serveur avec `Server.quit`

La `Routine` est lancée avec `Routine.play`.

2 Arbre sémantique

La construction de l'arbre sémantique associé à une expression suppose simplement de substituer aux éléments du langage leur constructeur de classe équivalent. Ces éléments sont :

- des espaces (ainsi que des tabulations et des retours à la ligne)
- des paires de parenthèses
- des mots-clef comme `lambda` ou `let`
- des symboles (éventuellement possédant un sens prédéfini)
- des constantes numériques

Les différents *lexèmes* (*tokens*) constituant le lexique seront tout d'abord identifiés par un analyseur lexical (*parser*) produit par `flex`, avant d'être passés à un analyseur syntaxique produit par `bison`, qui effectuera les substitutions nécessaires afin d'obtenir l'expression équivalente en SC.

Schématiquement, mis à part les formes spéciales `lambda`, `let`, `if`, et `module` (ainsi que leurs dérivées), qui devront faire l'objet d'un traitement spécial afin de séparer les séquences de paramètres des sous-expressions, les lambda-expressions entre parenthèses seront traduites par un même type de noeud `Call` qui contiendra des liens sur les sous-expressions (dont le sélecteur).

Les conversions sont données par le tableau suivant :

Expression en algèbre fonctionnelle	Constructeur SC correspondant
`(lambda (parm ...) body)`	`FLSC_Lambda([parm, ...], body)`
`(lambda (parm ... & rest) body)`	`FLSC_Lambda([parm, ..., rest], body, rest: true)`
`(let ((var def) ...) body)`	`FLSC_Let([var, ...], [def, ...], body)`
`(if test expr1 expr2)`	`FLSC_If(test, expr1, expr2)`
`(module (parm ...) fonct)`	`FLSC_Module([parm, ...], fonct)`
`(sel arg ...)`	`FLSC_Call(sel, [arg, ...])`
`ident`	`'ident' | FLSC_Var('ident')`
`num`	`FLSC_Num(num)`

Toutes ces classes héritent de la classe abstraite `FLSC_SemanticNode` afin d'assurer le polymorphisme dans l'arbre sémantique. Les classes qui définissent des fonctions (`FLSC_Lambda`, `FLSC_Module`) héritent également de la sous-classe `FLSC_FuncDef` qui possède certaines particularités qu'on verra ci-après.

Certaines constructions sont sémantiquement équivalentes :

La forme `let` possède une équivalence sémantique :

`(let ((var def) ...) body)` ↔

`((lambda (var ...) body) def ...)`

et donc `FLSC_Let([var, ...], [def, ...], body)` aura la même valeur que :

`FLSC_Call(FLSC_Lambda([var, ...], body), [def, ...]))`

Néanmoins, la forme `let` est traitée séparément pour des raisons d'optimisation.

3 Interpétation de l'arbre sémantique

L'interprétation sous la forme d'un enregistrement audionumérique se produit en deux temps qu'il est avantageux de dissocier :

1. Production d'une spécification d'enregistrement numérique, c'est à dire en SC une orchestration (ensemble de définitions de synthétiseurs ou `SynthDef`) ainsi qu'une partition (`Score`) qui est une séquence ordonnée de paquets OSC qui définissent les instanciations et libérations des synthétiseurs (`Synth`).
2. Rendu de la structure obtenue (par les méthodes de `SynthDef` et de `Score`) sous la forme d'un fichier audio ; le rendu pourra être effectué hors temps réel de façon à éviter toute surcharge de calcul (et inversement de façon à accélérer le processus lorsque la charge de calcul est faible).

La première phase est de loin la plus complexe de notre point de vue, puisque la deuxième est presque entièrement prise en charge par SuperCollider. Nous nous concentrerons donc sur ce premier point.

Nous nous contenterons ici de décrire le processus dans son principe général ; le code source correspondant est disponible sur GitHub [18] ; en outre, un manuel d'utilisation est donné en annexe (cf. A).

Globalement, l'interprétation d'un arbre sémantique consistera à appeler la méthode `FLSC_SemanticNode.value` à sa racine ; le résultat dépendra de l'interprétation récursive des sous-arbres.

3.1 Schéma d'évaluation global

Le schéma global des classes d'extension et de l'évaluation de la phase 1 est donné par la figure III.1 page suivante. Celui-ci est simplifié de façon à éviter de représenter certaines subtilités qui ne sont pas nécessaires à la compréhension générale du mécanisme.

Celle-ci se décompose en 3 phases :

1. Evaluation de l'arbre sémantique (`FLSC_SemanticNode`) pour obtenir une spécification des éléments qui composent la partition virtuelle (`FLSC_ScoreSpec`)
2. Evaluation de la `FLSC_ScoreSpec` pour obtenir une partition virtuelle (`FLSC_Score`)
3. Evaluation du `FLSC_Score` pour obtenir les éléments SC nécessaires au rendu audio (un `Set` de `SynthDef` et un `Score`)

3.2 Spécification des éléments

Le but final de l'évaluation de l'arbre sémantique est d'obtenir les définitions de synthétiseurs et la liste de paquets OSC qui définissent la partition virtuelle.

Cet objectif doit être atteint de façon indirecte, par des représentations intermédiaires, car un même noeud de l'arbre sémantique, suivant le contexte, peut donner lieu à un ou plusieurs groupes de paquets OSC : c'est

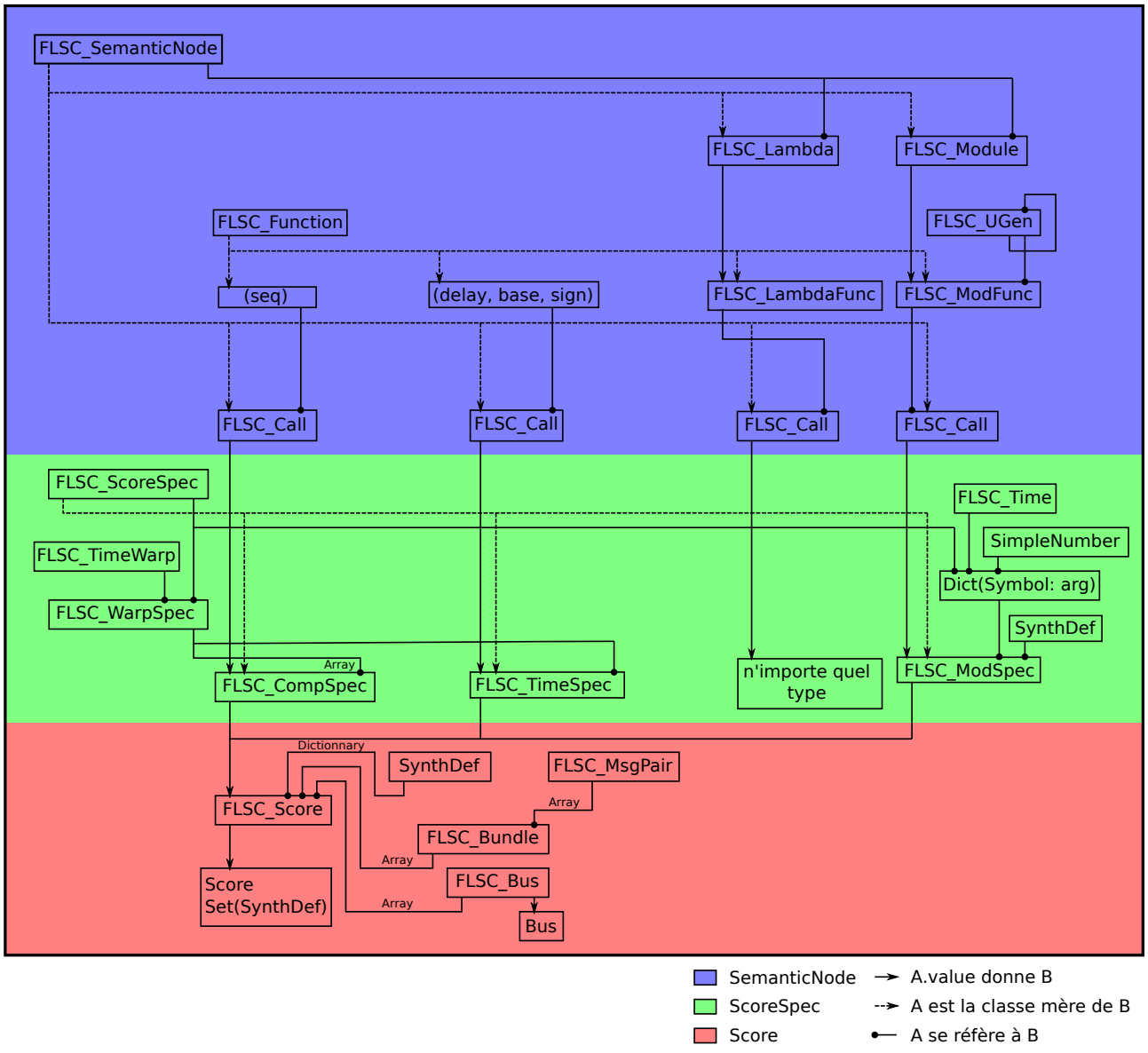

FIGURE III.1 – Schéma des classes et de l'évaluation

notamment le cas lorsqu'un même objet (fragment, signal ou contrôle) est lié a une variable et réutilisé plusieurs fois à des dates différentes.

On produira donc en premier lieu un objet `FLSC_ScoreSpec`, qui est d'un type récursif abstrait, et qui décrit le contenu (abstrait des transformations temporelles) des noeuds de l'arbre sémantique qui correspondent à des fragments ou à des objets temporels. Chaque nœud de cet arbre contiendra (dans le cas où il correspond à un module de synthèse) la spécification d'un ensemble de *paires* OSC (messages de début et de fin des modules de synthèse).

A noter qu'au cours de cette étape, les appels de fonction seront résolus ; cette représentation intermédiaire constitue donc un premier déroulement de l'arbre sémantique.

3.2.1 Résolution des appels de fonction

Les nœuds `FLSC_FuncDef` (qui correspondent aux formes spéciales `lambda` ou `module`) font appel au mécanisme des *fonctions* ; les nœuds `FLSC_Let`, dans lesquels cette fonction est appelée immédiatement, font appel à un mécanisme analogue mais plus compact.

En conséquence, `FLSC_FuncDef.value` retourne un objet du type `FLSC_Function`, qui peut occuper la place du sélecteur dans un nœud `FLSC_Call`. Ces objets, outre le sous-arbre qui définit leur corps, contiennent également un tableau qui décrit le nom de leurs paramètres, et qui sera utilisé lors de l'appel de la fonction pour effectuer les substitutions nécessaires.

Lors de l'appel par `FLSC_Call`, les arguments de l'appel sont associés dans l'ordre aux noms des paramètres, ce qui produit une structure `FLSC_Context` qui contient un `Dictionnary` des associations. Lorsqu'ensuite on rencontre dans le sous-arbre sémantique un nom de variable, on ira chercher sa signification dans ce dictionnaire.

Lorsque la `FLSC_Function` est créée avec le paramètre `rest: true`, les arguments supplémentaires à partir du dernier paramètre sont associés dans une liste au dernier paramètre de la fonction. Cela permet notamment la réalisation de fonctions telles que `list`.

Il est possible néanmoins que les noms de variable se réfèrent à un autre contexte plus général ; en conséquence, les objets `FLSC_Context` se réfèrent récursivement au contexte dans lequel ils sont définis (on a donc un arbre de `FLSC_Context` dont chaque nœud définit une table des symboles). Le mécanisme de *clôture lexicale* est mis en place en sauvegardant dans les objets `FLSC_Function` le contexte dans lequel elles ont été définies.

3.2.2 Evaluation par étapes

Certains des objets produits par l'évaluation de l'arbre sémantique sont *évalués par étapes* ; c'est à dire que l'évaluation se poursuivra dans une phase ultérieure du traitement.

C'est le cas pour les objets du type `FLSC_ScoreSpec` ou `FLSC_UGen`. En effet ces objets produisent d'autres objets au cours de leur évaluation, dont on ne souhaite avoir qu'une seule instance dans un même contexte (pour une question d'économie de ressources). On utilise dans ces cas un mécanisme de *fabrique* (*factory*), qui permet à un meta-objet de produire un autre objet à la demande.

Le premier type doit encore être évalué afin d'obtenir l'ensemble de messages OSC qui définissent la partition virtuelle, puisqu'il est possible qu'un seul objet produise plusieurs ensembles de messages. Lorsqu'il est présent plusieurs fois dans la même signature temporelle, on ne souhaite l'évaluer qu'une seule fois.

Les définitions de modules (du type `FLSC_ModFunc`), produites par `FLSC_Module.value`, sont également évaluées par étapes (elles contiennent des composantes du type `FLSC_UGen`) ; il faudra encore au moment de l'appel déterminer le type des composantes et de la sortie en fonction des types en entrée (constante, contrôle ou signal).

Dans tous les cas cela pose un problème : une même définition, si elle est réutilisée plusieurs fois dans le même contexte par l'intermédiaire d'une introduction de variable (donc un `FLSC_Call`), ne doit pas être réévaluée à chaque fois, faute de quoi des objets identiques seront produits.

On utilisera donc, lorsqu'un objet de ce type (`FLSC_ScoreSpec` ou `FLSC_UGen`) est lié à une variable, un objet spécial (respectivement de type `FLSC_VarSpec` ou `FLSC_VarUGen`) qui aura pour fonction de l'encapsuler et d'ajouter une référence dans le contexte qui permettra de ne l'évaluer qu'une seule fois lors de l'étape d'évaluation suivante.

Dans le cadre de l'évalutation successive de ces objets, on évaluera tout d'abord les objets référencés ; on passera dans les appels récursifs une variable spéciale `varDict` pour donner les références aux objets précalculés. Dans ce but, les objets de ce type contiennent la liste (ordonnée suivant l'ordre des références) des variables référencées, dans un champ `varList` dédié.

Les variables qui sont sujettes à ce type d'interprétation sont précédées, lors de leur déclaration, par la marque de *non-unicité* `!`. Il est de la responsabilité de l'utilisateur de les signaler explicitement, puisque le comportement souhaité est parfois ambigu (notamment dans le cas où l'objet en question possède un comportement stochastique : il est possible de réutiliser le même objet ou d'utiliser un objet à chaque fois différent).

3.2.3 Création des modules

En raison de l'impératif d'invariance algorithmique, `FLSC_Module.value` produit un objet `FLSC_Function` (du sous-type `FLSC_ModFunc`) qui est déjà partiellement traité ; on doit évaluer sous cette forme spéciale tout ce qui est de l'ordre d'un appel de fonction, et ne retenir que la définition d'un graphe de `UGen` (au type des paramètres près).

Cet objet aura pour but de produire, lors de son appel, le `FLSC_ScoreSpec` (du sous-type `FLSC_ModSpec`) correspondant, qui contiendra la `SynthDef` associée à cette instance du module.

La `SynthDef` en question est variante suivant les types de fonctions temporelles qu'elle véhicule (signaux, contrôles, constantes) ; cette `FLSC_Function` produira donc potentiellement, suivant ses instances, plusieurs `SynthDef` différentes, qui seront différenciées par un suffixe (déterminé par les types en entrée).

En outre les paramètres du module, dans les `SynthDef`, seront convertis au type `Control` (qui décrit un paramètre d'entrée d'un module de synthèse), et les substitutions correspondantes (par des nombres ou des indices de `Bus`) seront effectuées juste avant le rendu, par l'intermédiaire des paramètres des messages OSC correspondants.

FLSC_ModFunc

La `FLSC_ModFunc` est constituée des champs suivants :

`baseName :` le nom de base de la famille de SynthDef associée

`args :` le nom de ses arguments

`varList :` la liste des variables employées dans le module

`uGenGraph :` le graphe de ses composantes

Le `baseName` est un symbole unique obentu par le générateur d'UID (*unique identifier*) de SC, et les `args` sont simplement les noms de variables qui définissent le `FLSC_Module`.

Le `uGenGraph` est obtenu par l'évaluation récursive du corps du module, qui est du type récursif `FLSC_UGen`. En effet, les `FLSC_UGen` se réfèrent aux autres objets qu'ils prennent en entrée.

Ce type est celui qui est retourné par les `FLSC_Function` qui correspondent aux primitives de fonctions temporelles de l'expression fonctionnelle. Il est également retourné par les `FLSC_Function` qui représentent les opérations algébriques, lorsqu'au moins un de leurs arguments est du type `FLSC_UGen` ; l'identification du type est donnée par la méthode `.rate` (méthode polymorphique qui sert à distinguer les différentes bases de temps des objets SC) : `SimpleNumber.rate` retourne `'scalar'` alors que `FLSC_UGen.rate` retourne `'temporal'`.

Au cours de cette évaluation, tous les appels de fonction sont résolus (`let`, appels à des `lambda`, itérations, etc...). Les éléments terminaux sont soit des nombres, soit des `FLSC_UGen`.

Il peut s'agir de `FLSC_Control`, qui est le sous-type qui est substitué aux arguments du module. Lors de l'évaluation de `FLSC_Module`, on associe dans un `FLSC_Context` (qu'on créé pour l'occasion, comme si il s'agissait d'un appel de fonction) les noms des arguments (`args`) avec les objets `FLSC_Control` (encapsulés par un `FLSC_VarUGen`) qu'on crée pour les représenter, de façon à pouvoir effectuer la substitution des variables correspondantes. Les autres variables seront recherchées récursivement dans le contexte du `FLSC_Module`.

Lorqu'on rencontre un `FLSC_VarUGen` (une variable qui représente un `FLSC_UGen`, et qui peut ou non être un `FLSC_Control`), la branche d'évaluation s'arrête (cette variable a déjà été évaluée dans son contexte propre). On ajoute alors l'objet correspondant, ainsi que les variables qui sont référencées par lui, dans la `varList`.

FLSC_ModSpec

`FLSC_ModFunc.value` retourne un objet du type `FLSC_ModSpec`. Il s'agit de produire, lorsque le module est instancié, la `SynthDef` correspondante, ainsi que ses arguments (qui peuvent correspondre à des sous-spécifications).

Le graphe de `UGen` qui compose la `SynthDef` est obtenu par l'appel récursif `FLSC_UGen.value`. Celui-ci retourne le `UGen` approprié, suivant la base temporelle choisie en fonction des types entrants. Ceux-ci sont obtenus par la méthode `UGen.rate`, qui retourne soit `'control'` soit `'audio'` ; `SimpleNumber.rate`, quant à elle, retourne `'scalar'` ce qui permet de distinguer les constantes. Chaque `FLSC_UGen` effectue une opération de type `max` pour déterminer son type de sortie. Finalement, on retombe sur les éléments passés en argument au module (des `FLSC_ScoreSpec` ou des nombres, qui ont tous deux également la méthode `.rate`).

Comme tout appel de fonction, `FLSC_Call(FLSC_ModFunc, ...)` engendre la création d'un `FLSC_Context` qui associe dans un dictionnaire ses paramètres à leur valeur dans l'appel. On l'utilisera pour déterminer le type (scalaire, contrôle, audio) des `FLSC_Control` invoqués, au travers de la méthode `.rate`.

`FLSC_UGen.value` est appelée avec un paramètre supplémentaire `varDict` dérivé de `FLSC_ModFunc.varList`, qui sert à noter les instanciations des objets référencés par des `FLSC_VarUGen` de façon à les réutiliser lorsqu'elles apparaissent plusieurs fois.

On itère donc tout d'abord sur `FLSC_ModFunc.varList` pour obtenir un `IdentityDictionnary` qui associe ces `FLSC_UGen` à la valeur obtenue par leur évaluation ; cela entraîne la création des `Control` dont dépend la `SynthDef` . C'est cet objet qui sera passé en argument à `FLSC_UGen.value`.

Lorsqu'on rencontre un `FLSC_VarUGen`, on recherche dans `varDict` l'objet référencé par la variable et on substitue la valeur précalculée.

Il y a un troisième paramètre `timeControls`, du type `Dictionnary`, qui référence les `Control` de type temporel créés par les primitives d'enveloppe (`env` et `seg`). En effet les classes d'enveloppe en SC sont paramétrées par des listes d'intervalles de temps, alors que nous utilisons des indices de signature pour des raisons de cohérence. Il faut donc ajouter des paramètres cachés à la `SynthDef`, qui permettront de passer les arguments temporels lors de l'instanciation du module.

Les modules utilisant des références temporelles ajoutent donc dans `timeControls` une association de l'UID qui caractérise une `FLSC_TimeUGen` donnée à l'objet qui décrit l'ensemble d'intervalles (objet `FLSC_Time` créé par celle-ci). L'objet `timeControls` sera concaténé à l'issue de l'évaluation aux arguments de la `FLSC_ModSpec`.

Lorsqu'on a obtenu le graphe de `UGen` complet, on appelle `UGen.rate` pour déterminer son type de sortie ; en conséquence de quoi on ajoute l'objet `Out.ar` ou `Out.kr` correspondant à la racine du graphe. Dans l'éventualité où le type de sortie serait `'scalar'` (un nombre), on retourne simplement ce nombre.

Le `Bus` de sortie est déterminé par un autre paramètre caché `out` ; il sera calculé lors de l'évaluation de la `FLSC_ModSpec`.

3.2.4 Création des éléments de spécification

Les éléments de la spécification de partition, du type abstrait `FLSC_ScoreSpec`, sont déclinés dans les types concrets suivants :

`FLSC_ModSpec :` les modules standard

`FLSC_TimeSpec :` les formes d'altération de la temporalité (`delay`, `base` et `sign`)

`FLSC_CompSpec :` la composition séquentielle `seq`

`FLSC_ListSpec :` la composition parallèle, donnée par une liste récursive d'objets

`FLSC_VarSpec :` les éléments produits par l'appel à une variable, qui doivent être traités d'une façon particulière en raison de l'évaluation par étapes.

Ils sont constitués des éléments suivants :

`rate :` un type de sortie (signal, contrôle ou constante)

`varList :` la liste des variables référencées par des `FLSC_VarSpec` dans le même contexte de signature temporelle

contenu **:** la définition du contenu du nœud (suivant le type concret) :

`FLSC_ModSpec :` une paire OSC spécifiée par :

`def :` une `SynthDef` obtenue par appel à la fabrique (suivant les types en entrée)

`args :` un `Dictionnary` qui aux paramètres de la `SynthDef` associe l'élément correspondant : soit une constante (paramètre numérique), soit un `FLSC_Time` (paramètre dépendant de la signature temporelle), soit une `FLSC_ScoreSpec` (signal ou contrôle en entrée)

`FLSC_CompSpec : subSpecs :` une liste de `FLSC_WarpSpec`

`FLSC_TimeSpec : subSpec :` une `FLSC_WarpSpec`

`FLSC_VarSpec: subSpec: une FLSC_ScoreSpec`

`FLSC_ListSpec : subSpecs :` une liste de `FLSC_ListSpec` ou `FLSC_ScoreSpec`

Les types auquels il est fait référence :

- `FLSC_Time`
- `FLSC_WarpSpec`

et par suite le type `FLSC_TimeWarp` (qui est un type composante de `FLSC_WarpSpec`) sont définis de la façon suivante :

`FLSC_Time` : ce type décrit une liste d'intervalles de temps, et contient simplement les indices correspondants dans la signature temporelle locale ; ce type est nécessaire car la méthode qui permet par la suite de l'évaluer entraîne sa transformation par les opérations sur la signature temporelle (il faut donc le distinguer des autres types de paramètres)

`FLSC_WarpSpec` : ce type héberge les spécifications d'une partition d'ordre inférieur, lorsqu'il y a une altération de la signature temporelle. Il contient :

`timeWarp` : une modification de la signature temporelle : une `FLSC_TimeWarp`

`subSpec` : une spécification de partition d'ordre inférieur : une `FLSC_ScoreSpec`

`varList` : la liste des variables référencées par des `FLSC_VarSpec` dans cette sous-spécification

`FLSC_TimeWarp` : ce type, systématiquement associé à une sous-spécification, est une fonction essentiellement numérique qui à un élément de la signature temporelle (indice ou date) de l'appelé associe l'élément correspondant dans le contexte de l'appelant. Cette fonction admet également un élément spécial en entrée, le `Symbol 'end'`, qui correspond à la fin du support temporel de l'appelé. Elle peut retourner `'end'` pour signifier la fin du support temporel de l'appelant. Les modules standard ont implicitement pour support temporel `[0, 'end']`.

Les éléments de l'arbre sémantique sont traités récursivement ; les `FLSC_Call` évaluent en premier lieu leurs arguments, avant d'appeler la `FLSC_Function` correspondante.

Pour les nœuds qui retournent des `FLSC_ScoreSpec`, le type de sortie de celles-ci est suffisant pour déterminer les paramètres de la fonction appelante (le type des `Bus` d'entrée permet de déterminer son type de sortie, ainsi que sa `SynthDef`, si il y a lieu). Quant aux autres, après résolution des appels de fonction, ils ne peuvent retourner que des constantes.

Des fonctions particulières de type `FLSC_Function` sont mises en place pour gérer les compositions (parallèle et `seq`) et les modifications de signature (`delay`, `base`, `sign`).

Toute expression qui décrit un fragment doit produire une liste récursive de `FLSC_ScoreSpec` en retour de l'évaluation de son arbre sémantique. Lorsqu'un objet temporel est attendu, les listes sont interprétées comme des `FLSC_ListSpec`.

3.3 Création d'une partition virtuelle

A partir de l'objet `FLSC_ScoreSpec` racine, on souhaitera ensuite obtenir une partition virtuelle SC (c'est à dire un `Score` et une liste de `SynthDef`).

Il faudra déterminer les instances individuelles des `FLSC_ScoreSpec` qui doivent être produites (un même `FLSC_ScoreSpec` peut produire plusieurs instances, si il est appelé plusieurs fois dans des signatures temporelles différentes). Lorsque ces instances produisent des modules, les messages de démarrage et d'arrêt de ceux-ci seront représentés par des objets `FLSC_MsgPair`, qui seront regroupés (par support temporel) dans des `FLSC_Bundle`.

Pour chaque instance il faudra employer un mécanisme d'allocation de `Bus` (afin de réutiliser ceux-ci autant que possible, puisqu'ils sont en nombre limité). On représentera temporairement les `Bus` par des objets `FLSC_Bus`.

Il faudra également déterminer l'ensemble des `SynthDef` nécessaires à la réalisation de la partition.

Pour tout cela on fera appel à la méthode `FLSC_ScoreSpec.value`, sur cet objet racine ; elle sera rappelée récursivement sur les `FLSC_ScoreSpec` d'ordre inférieur.

3.3.1 FLSC_Score

Cette méthode retourne un objet `FLSC_Score`, dont les champs sont :

`outBus` : le `FLSC_Bus` de sortie finalement choisi par l'objet

`defs` : un dictionnaire (`Dictionnary`) d'associations (nom : `SynthDef`), qui correspond à celles qui sont utilisées dans ce sous-arbre

`busList` : une liste de `FLSC_Bus`, qui représente ceux qui sont utilisés dans ce sous-arbre

`bundle` : une liste de `FLSC_MsgPair` qui représente le contenu du support temporel local

`bundleList` : une liste de `FLSC_Bundle` qui représentent le contenu de tous les sous-supports des sous-arbres

La méthode `FLSC_ScoreSpec.value` est appelée avec trois paramètres :

`outBus` : le `FLSC_Bus` de sortie demandé par l'appelant (si c'est une composition ou une altération de signature), ou `nil` (si c'est un module).

`timeWarp :` la liste des fonctions de redéfinition de la signature temporelle accumulée au dessus de ce sous-arbre.

`varDict :` un `IdentityDictionnary` qui contient des associations `(FLSC_ScoreSpec -> FLSC_Bus)` ; les clefs de ce dictionnaire sont les `FLSC_ScoreSpec` référencées par des variables (`FLSC_VarSpec`) et qui sont donc regroupées dans `FLSC_WarpSpec.varList`. Celles-ci sont instanciées à l'entrée dans la signature temporelle locale, pour ce sous-arbre, et n'ont donc pas besoin d'être instanciées à nouveau.

A la racine, on appelle `FLSC_ScoreSpec.value(systemOut, [], IdentityDictionnary([]))` ; `systemOut` est une variable système qui renvoie à la sortie audio de SC, `[]` est la liste vide de fonctions (l'identité), `IdentityDictionnary([])` est un dictionnaire vide.

3.3.2 Détermination des paquets OSC

Si la `FLSC_ScoreSpec` représente un module, la `FLSC_MsgPair` correspondante est construite de la façon suivante : on construit la paire OSC (essentiellement le message de début, le message de fin étant toujours le même) à partir du nom de la `SynthDef` et de ses paramètres, et de la définition de l'élément de spécification.

On itère donc sur les paramètres `FLSC_ModSpec.args[]` (qui sont hétérogènes) ; les nombres sont laissés tels qu'ils sont et les `FLSC_Time` sont remplacés par la valeur donnée par :

```
var warpTimes = FLSC_Time.times.collect({|item|
        timeWarp.inject(item, {|val, func|
                func.value(val);
        });
});
[warpTimes[0..(warpTimes.size-2)],
 warpTimes[1..(warpTimes.size-1)]].flop.collect({|item|
        item[1] - item[0]});
```

qui effectue la différence (après application de la distorsion temporelle) entre tous les indices successifs du `FLSC_Time`, ce qui résulte en la liste d'intervalles souhaitée.

Pour ce qui est des `FLSC_ScoreSpec`, la procédure résulte en un appel récursif.

On rappelle récursivement avec :

1. `nil` comme premier paramètre
2. `timeWarp` (passé en argument) comme deuxième paramètre
3. `varDict` (passé en argument) comme troisième paramètre

A partir des objets `FLSC_Score` résultants de ces appels, on ajoute les définitions dans `defs`, les bus dans `busList`, on concatène les `bundle` et le bundle qu'on a accumulé jusque là, et de même pour les `bundleList`.

Si le `outBus` reçu n'est pas `nil`, il est choisi comme sortie ; sinon on en créée un, et il est ajouté dans `busList`. Le `outBus` retourné est choisi comme paramètre `out` dans les arguments du module.

La `FLSC_MsgPair` obtenu (à partir du nom de la `SynthDef`, du nom de ses arguments, et des appels récursifs) est ajoutée au début du `bundle` accumulé ; la `SynthDef` est ajoutée à `defs`. Puis l'objet `FLSC_Score` ainsi obtenu constitue la valeur retournée.

3.3.3 Variables

Si la sous-spécification est une `FLSC_VarSpec`, elle recherche sa référence dans `varDict`. Elle se retourne simplement le `FLSC_Score` correspondant.

Dans le cas où le `outBus` reçu n'est pas `nil`, elle créé un *pipe* (tuyau) qui consiste en un module élémentaire qui envoie le bus d'entrée sur le bus de sortie. Cela permet de respecter les deux contraintes (le bus d'entrée est celui passé en argument, et le bus de sortie est le bus préexistant). Cela implique néanmoins la création d'un `FLSC_MsgPair`, qui est ajouté en tête du `bundle`.

3.3.4 Altérations de temporalité

Les `FLSC_CompSpec` et `FLSC_SignSpec` se comportent d'une façon légèrement différente ; dans ce cas il n'y a pas de création d'une paire OSC (il n'y a pas de module de synthèse associé).

Pour chaque sous-spécification (il y en a soit une, soit plusieurs), on rappelle avec les paramètres suivants :

1. `outBus` (les bus d'entrée sont concentrés sur le bus de sortie pour les compositions)
2. `[FLSC_WarpSpec.timeWarp] ++ timeWarp` (cette méthode produit la liste de `FLSC_TimeWarp` constituée par la composition de la distorsion locale et de la sous-distorsion)

3. un `IdentityDictionnary` créé par itération sur la `varList`, afin d'attribuer une valeur à chaque objet référencé par une variable.

Le `bundle` est transformé en `FLSC_Bundle` par l'ajout des dates de début et de fin, qui sont obtenues respectivement par les formules

`timeWarp.inject(0, {|val, func| func.value(val)})` et

`timeWarp.inject('end', {|val, func| func.value(val)})`,

puis cet objet est ajouté à la `bundleList` accumulée.

Les définitions sont ajoutées dans `defs`, les bus dans `busList`, le `bundle` est vide, et le `outBus` est celui demandé (ou bien il est créé si aucun n'est demandé).

Dans l'éventualité où une sous-spécification est appelée sans changement de signature temporelle, on préserve `timeWarp` et `varDict` dans l'appel, et le `bundle` est accumulé dans le `bundle` de sortie comme pour `FLSC_ModSpec`.

3.3.5 Composition parallèle

Le comportement de `FLSC_ListSpec` est similaire à celui des altérations de signature temporelle, mis à part que celle-ci ne change pas. On préserve donc `outBus`, `timeWarp` et `varDict` dans l'appel récursif aux éléments de la liste. Les définitions sont ajoutées dans `defs`, les bus dans `busList`, les `bundleList` sont concaténées. Le `outBus` est celui demandé, et le `bundle` devrait être vide, puisqu'on doit être au niveau des fragments.

3.3.6 Attribution des Bus et substitution

A partir du `FLSC_Score.busList` renvoyé par l'arbre de `FLSC_ScoreSpec`, il est possible d'attribuer le nombre minimal de `Bus` nécessaire. Il faut pour cela construire un double index (sur les champs `start` et `end` de `FLSC_Bus`) ; ensuite on peut itérer sur ces deux index en parallèle (en avançant à chaque tour d'un côté ou de l'autre, suivant la date la plus faible). Avant d'avancer sur un index, on résout l'évenement concerné :

- si c'est un début, on attribue un `Bus` au `FLSC_Bus` en question (soit en en prenant un dans la réserve, soit en en créant un nouveau si la réserve est vide)
- si c'est une fin, le `Bus` du `FLSC_Bus` est replacé dans la réserve

Il faut distinguer les bus audio et de contrôle par l'attribut `FLSC_Bus.type`, et donc utiliser deux réserves séparées.

A l'issue de ce processus, tous les `FLSC_Bus` se sont vus attribuer un `Bus` qui n'est pas utilisé ailleurs au même moment.

On peut donc effectuer les substitutions dans les `FLSC_MsgPair` pour obtenir les messages et paquets OSC proprement dits. On itère sur chaque `FLSC_Bundle` dans l'ordre, en créant en parallèle le paquet de début et le paquet de fin (avec leurs dates respectives). `FLSC_MsgPair.value` renvoie un couple de messages construits au moyen de `Server.makeBundle`, l'un contenant `Synth(FLSC_MsgPair.defName, FLSC_MsgPair.args)` (qu'on affecte à une variable `synth`) et l'autre contenant `synth.free`. La concaténation des deux dates et des deux séries de messages produit les deux paquets OSC souhaités. La concaténation de tous les paquets permet de construire l'objet `Score` recherché.

3.4 Rendu audionumérique

Le rendu peut être effectué en temps réel comme dans l'exemple (cf 1.4 page 45), ou bien hors temps réel, au moyen de la méthode `Score.recordNRT`. Dans tous les cas il faut au préalable charger les `SynthDef` en itérant sur la liste avec `SynthDef.add` ; il faut attendre après cette opération le message de synchronisation du serveur.

4 Conclusion

Nous avons montré, d'un bout à l'autre, comment une expression FLSC valide peut être réalisée, de façon automatique, sous la forme d'un enregistrement audionumérique. Ce langage présente l'avantage d'être à la fois très flexible, reposant sur des objets élémentaires et néanmoins pertinents du point de vue de l'audition humaine, et dans le même temps de prendre en charge implicitement les aspects techniques de la synthèse sonore, permettant à l'utilisateur de programmer à un niveau très abstrait.

A l'heure actuelle, ce langage reste centré sur la production d'enregistrements hors temps réel, mais des extensions sont envisageables afin de le rendre réactif et non-déterministe, par l'introduction d'un mécanisme de

calcul différé, de déclencheurs, et de modules d'entrée accessibles par des interfaces courantes. Il est également assez peu optimal dans sa réalisation, étant donné que cela ne figurait pas parmi nos priorités – mais sans nul doute de nombreuses améliorations pourraient être effectuées afin d'en augmenter l'efficacité.

Nous utiliserons par la suite cette implémentation afin de produire les enregistrements associés aux structures musicales étudiées, notamment dans le but d'évaluer la pertinence de nos hypothèses en confrontation directe avec la perception humaine. Il sera également possible, à l'issue de ces vérifications, de réimplémenter dans des langages plus classiques les schémas jugés pertinents, étant donné que nous pouvons les décrire de façon exacte.

Chapitre IV

Modélisation des objets Schaefferiens

La suite de notre travail consiste à rechercher des hypothèses vraisemblables sur les catégories qui apparaissent dans la perception humaine du son ; il s'agit d'une étape indispensable afin de pouvoir qualifier les éléments d'une composition musicale, et ainsi pouvoir les décrire par une notation symbolique (qu'il s'agisse de noms ou de symboles graphiques).

Nous procèderons donc à la formulation d'un système complet de synthèse sonore, inspiré par la théorie Schaefferienne et ses développements, notamment ceux de Jean-Louis Di Santo avec lequel nous avons travaillé. Pour cela nous commencerons par proposer une structure mathématique pour ce système (1), puis nous en effectuerons la traduction en FLSC (2) ; enfin nous ferons l'inventaire des éléments que nous avons pu identifier, et qui semblent pertinents au sein de cette structure (3).

1 Expression générale des objets Shaefferiens

On donne ici la forme générale des objets audionumériques définis selon des critères Schaefferiens. Cette définition repose sur un certain nombre de fonctions abstraites (dont on précise le type) ; l'ensemble des combinaisons d'instances de ces fonctions, dont nous donnerons certaines définitions possibles, délimite l'espace expressif décrit par cette famille d'objets audionumériques.

Ces fonctions sont définies successivement, avant d'aboutir à leur synthèse dans une expression globale (IV.1).

1.1 Dynamique

La *dynamique* (appelée également le *profil dynamique*) correspond à l'évolution de l'énergie globale de l'objet au cours du temps. On la décrit par une fonction temporelle, le gain de base d^P, qui associe un coefficient de gain à tout élément de la base de temps ; ce coefficient multiplicatif sera appliqué à l'objet dans son ensemble.

Le gain de base doit être nul au début et à la fin du support temporel de l'objet audionumérique, et non nul sur au moins une partie de ce dernier ; en effet c'est ce critère qui constitue la présence ou l'absence de l'objet. En conséquence il s'agit nécessairement d'un profil, c'est pourquoi il sera noté d^P.

■ $d^P : \mathbb{T} \to \mathbb{R}^+$: une fonction temporelle

1.2 Profil mélodique

Le *profil mélodique* correspond à l'évolution globale de la hauteur de l'objet au cours du temps. Il est possible que la hauteur soit constante, dans ce cas on la décrit par la fréquence de base f_0. Si elle est dynamique, on la décrit par une fonction temporelle f_0^P, le profil de fréquence de base, qui associe une fréquence à tout élément de la base de temps.

La fréquence de base sera employée comme référence dans le calcul de la fréquence des composantes individuelles.

■ $f_0 : \mathbb{R}^+$: un nombre

▶ $f_0^P : \mathbb{R}^{+P} = \mathbb{T} \to \mathbb{R}^+$: une fonction temporelle

Dans la suite on ne considérera que le cas général où la hauteur est dynamique, puisqu'on peut modéliser le cas où elle est statique en posant $\forall t,\ f_0^P(t) = f_0$.

1.3 Calibre

Le *calibre* est la largeur de la bande spectrale occupée par l'objet. Celle-ci sera décrite relativement à la fréquence de base, au travers de la *fréquence relative limite* $n \in \mathbb{N}^*$, qui correspond (approximativement) au multiple maximum de la fréquence de base f_0 (1.2) qui sera employé comme fréquence des composantes de l'objet.

Comme les modifications de la composition spectrale peuvent entraîner l'apparition de composantes supplémentaires, il sera nécessaire de calculer le nombre de composantes par la *fonction de remplissage* N ; celle-ci dépendra bien entendu des transformations effectuées. Le nombre de composantes sera donc $N(n)$.

■ $n : \mathbb{N}^*$: un entier positif

■ $N : \mathbb{N}^* \to \mathbb{N}^*$: une fonction algébrique

1.4 Masse

La notion Schaefferienne de *masse* établit une classification typologique des matériaux sonores. Nous retiendrons l'idée qu'une masse peut être tonique (harmonique) ou complexe (inharmonique et/ou bruitée). Ces variations seront décrites en 3.2 page 65.

On peut décrire la masse comme une fonction M qui, à chaque indice de composante (éventuellement bruité), associe le coefficient qui relie la fréquence de cette composante à la fréquence de base. Ainsi cette fonction exprime, relativement à la fréquence de base, la structure spectrale du matériau sonore ; il s'agit d'une fonction de distorsion (*warping*) harmonique.

1.4.1 Cas statique

Dans le cas où la masse est invariante au cours du temps :

■ $M : \mathbb{N}^* \to \mathbb{R}^+$: une fonction algébrique

Cette fonction permet alors de calculer la fréquence relative $M(i)$ des composantes à partir de l'indice de composante $i \in \{1, \ldots, N(n)\}$. On note qu'un spectre harmonique correspond au cas où $M(i) = i$.

1.4.2 Fréquence absolue

La fréquence absolue des composantes peut alors être calculée à partir des fréquences relatives et du profil de fréquence de base $f_0^P(t)$ (1.2) :

▶ $f(i)(t) = M(i) \cdot f_0^P(t)$

Si la fréquence de base et la masse (cf. 1.4.4) sont fixes, on peut substituer $f(i) = M(i) \cdot f_0$ à $f(i)(t)$.

1.4.3 Phase initiale

La caractérisation du signal associé à chaque composante doit être complétée par une information de phase initiale. Une modulation de phase continue étant équivalente à une modulation de fréquence, on travaille ici dans l'hypothèse d'une phase initiale constante $\phi(i)$:

■ $\phi : \mathbb{N}^* \to [-\pi, \pi]$: une fonction algébrique

1.4.4 Profil de masse

Dans le cas général, la structure spectrale peut être variante au cours du temps : les coefficients de masse peuvent varier. Alors, les paramètres qui définissent la masse sont des fonctions temporelles. On devra alors exprimer la masse comme la transposition dans le domaine temporel de M, le profil de masse $M^\#$ (cf. Opérations algébriques sur les fonctions temporelles) :

■ $M^\# : \mathbb{N}^* \to (\mathbb{T} \to \mathbb{R}^+)$: une fonctionnelle temporelle

Comme on l'a vu il est possible d'omettre l'élément de notation $\#$, on substitue donc dans les équations $M(t)$ à M.

1.5 Critère harmonique

Le critère harmonique correspond aux notions Schaefferiennes de *timbre harmonique* et de *profil harmonique* ; il décrit l'enveloppe spectrale (éventuellement dynamique) de l'objet audionumérique.

On le représente par une fonction H qui associe à la fréquence d'une composante son gain relatif ; cette fréquence peut être donnée de façon absolue ou relativement à la fréquence de base de l'objet. Il est potentiellement pertinent de superposer ces deux phénomènes.
En pratique il est plus raisonnable de normaliser les gains relatifs pour obtenir une moyenne unitaire, de façon à conserver une relation claire entre la dynamique et l'énergie globale.

1.5.1 Timbre harmonique

Le timbre harmonique est une fonction algébrique indépendante du temps.
Dans le cas où les fréquences sont prises de façon absolue, la fonction d'enveloppe spectrale absolue H_A associe un gain à la fréquence absolue $f(i)(t)$ (1.4.2) :

■ $H_A : \mathbb{R}^+ \to \mathbb{R}^+$: une fonction algébrique

Dans le cas où les fréquences sont prises de façon relative, la fonction d'enveloppe spectrale relative H_R associe un gain à l'indice de composante i ; dans ce calcul intervient la fréquence relative de base $M'(i)$, la fonction de calibre dynamique C et éventuellement la fonction de mesure du bruitage B. La fonction M' peut être différente de M car dans certains cas on ne souhaite garder des transformations du materiau que la partie qui est fondamentale à la situation des composantes dans le spectre (cf. 3.2 page 65).

■ $M' : \mathbb{N}^* \to \mathbb{R}^+$: une fonction algébrique

■ $C : \mathbb{N}^* \to \mathbb{R}^+$: une fonction algébrique

■ $B : \mathbb{N}^* \to (\mathbb{T} \to \mathbb{R}^+)$: une fonctionnelle temporelle

■ $H_R : \mathbb{N}^* \to \mathbb{R}^+$: une fonction algébrique

La fonction de timbre harmonique est alors calculée de la façon suivante, à partir de ces deux fonctions :

▶ $H(i)(t) = H_R(i) \cdot H_A(f(i)(t))$

L'un ou l'autre de ces comportements peut être neutralisé lorsque $\forall f : H_A(f) = 1$ ou $\forall i : H_R(i) = 1$.
L'amplitude des composantes est ensuite calculée suivant cette fonction et le gain global $d^P(t)$ (1.1) :

▶ $A(i)(t) = d^P(t) \cdot H(i)(t)$

1.5.2 Profil harmonique

Dans le cas général où l'enveloppe spectrale est elle-même dynamique, on la décrira par les tranpositions dans le domaine temporel de H_A et H_R, le profil spectral absolu $H_A^\#$ et le profil spectral relatif $H_R^\#$ (cf. Opérations algébriques sur les fonctions temporelles) :

■ $H_A^\# : \mathbb{R}^+ \to (\mathbb{T} \to \mathbb{R}^+)$: une fonctionnelle temporelle

■ $H_R^\# : \mathbb{N}^* \to (\mathbb{T} \to \mathbb{R}^+)$: une fonctionnelle temporelle

On substitue alors dans les équations à H_A et H_R, respectivement $H_A(t)$ et $H_R(t)$, en omettant le $\#$.

1.6 Forme générale

Les générateurs de signaux employés sont tous de type sinusoïdal, ils sont définis par la fonction *osc* (cf. Générateur élémentaire de signal), sur un support temporel hypothétique $T = [T_0, T_f]$.

■ $osc : \begin{cases} (\mathbb{T} \to \mathbb{R}^+) \times [-\pi, \pi] \times \mathbb{T} \to \mathbb{R} \\ f, \phi, t \mapsto sin\left(\int_{T_0}^{t} f(\tau) \cdot d\tau + \phi \right) \end{cases}$

Le fragment audionumérique associé à un objet défini de façon Schaefferienne, et restitué avec n composantes est alors défini comme un signal s :

▶ $s(t) = \sum_{i=1}^{N} A(i)(t) \cdot osc(f(i)(t), \phi(i), t)$

1.6.1 Expression complète de la forme générale

On va tout d'abord rassembler les fonctions (éventuellement temporalisées) et scalaires qui définissent un objet audionumérique dans un tuple G ; on notera $\mathcal{G}$ l'ensemble des tuples de ce type.

La forme générale d'un objet audionumérique Schaefferien $s(t)$ s'exprime à partir du tuple G, de la façon donnée en fig. IV.1 page suivante.

Dans le cas où la fréquence de base, la masse et/ou le profil harmonique sont statiques, on peut substituer à $f_0^P(t)$, $M(i)(t)$, $f(i)(t)$, $H_R(i)(t)$, $H_A(i)(t)$ respectivement f_0, $M(i)$, $f(i)$, $H_R(i)$, $H_A(i)$.

$$
\begin{aligned}
&\textit{pour } G = \Big(L, (T_l)_L, d^P, f_0^P, n, N, M, \phi, H_A, H_R\Big) : \\
&\qquad \textit{soit } f(i)(t) = f_0^P(t) \cdot M(i)(t) : \\
&\forall t \in [T_0, T_L] : s(t) = \sum_{i=1}^{N(n)} \Bigg[d^P(t) \cdot H_R(i)(t) \cdot H_A\big(f(i)(t)\big)(t) \times \\
&\qquad\qquad \sin\left(\int_{T_0}^{t} f(i)(\tau) \cdot d\tau + \phi(i)\right)\Bigg]
\end{aligned}
$$

FIGURE IV.1 – Expression mathématique des objets Shaefferiens

2 Transcription des instruments Schaefferiens

L'expression de la fig. IV.1 peut être exprimée au moyen des éléments fonctionnels pour la synthèse sonore décrits en II page 22. On peut dans ce cas parler d'un *instrument virtuel* puisque les individus parmi cette famille d'objets audionumériques peuvent être produits par un appel de fonction.

Cette fonction prend de nombreux paramètres, dont certains sont d'un type complexe (fonctions d'ordre supérieur). Grâce à l'approche modulaire de l'algèbre fonctionnelle employée, il est néanmoins possible d'exprimer cette famille d'objets de façon univoque, à partir de modules algorithmiquement invariants.

2.1 Algorithme

L'algorithme IV.1 page suivante donne l'expression en synthèse modulaire FLSC de l'objet S associé à s. Cette expression est liée à une variable `instrsch` dans ce paquetage qui pourra être associé aux programmes s'y référant.

On commence par appliquer le support temporel $[0, T_f]$ par l'expression `(base [0 end] ...)`.

On calcule ensuite par `(let* (...) ...)` :

- les indices $\{1, \ldots, N(n)\}$: `(indexes (range 1 (nfunc cal)))`
- les phases $\phi(i)$: `(phases (map phi indexes))`
- les fréquences relatives $M(i)$: `(frel (map mass indexes))`
- les fréquences absolues $f(i)$: `(fabs (m* amp lfrel))` ; noter ici qu'en FLSC, le produit d'une liste par un objet quelconque est la liste des produits des éléments par l'objet

L'expression retombe sur la partie centrale de la formule :

$$\sum_{i=1}^{N(n)} \left[d^P(t) \cdot H_R(i)(t) \cdot H_A\big(f(i)(t)\big)(t) \times \sin\left(\int_{T_0}^{t} f(i)(\tau) \cdot d\tau + \phi(i)\right)\right]$$

Celle-ci est traduite par un produit de signaux itéré sur les listes à $N(n)$ éléments :

```
(map (lambda (ind fabs phase) (mn* ...)) indexes lfabs phases)
```

- $d^P(t)$ correspond à `amp`
- $H_R(i)(t)$ correspond à `(harm ind)`
- $H_A\big(f(i)(t)\big)(t)$ correspond à `(col fabs)`
- $\sin\left(\int_{T_0}^{t} f(i)(\tau) \cdot d\tau + \phi(i)\right)$ est traduit par `(osc fabs phase)`

L'expression mathématique qui est ainsi transcrite est équivalente à celle de S, à ceci près que le début du support T_0 est 0 (il peut être ajusté ensuite par des retards).

2.2 Composition

La composition par superposition ou juxtaposition d'un ensemble de fragments fait appel à la fonction de retard *ret* (cf. Retard) et à la composition parallèle *par* (cf. Superposition).

Alors, si E est l'ensemble des couples date-objet $e_i = (\theta_i, x_i^O)$, $i \in \{1, \ldots, N\}$ appartenant à une composition (ou à une sous-partie de cette composition), l'objet audionumérique y^O associé à E sera représenté comme une composition parallèle de retards sur les objets :

▶ $y^O = par\big(ret_{\theta_1}(x_1^O), \ldots, ret_{\theta_N}(x_N^O)\big)$

Ceci se traduirait en FLSC par une expression comme :

```
[(delay t1 x1) ... (delay tn xn)]
```

Algorithme IV.1 Expression de l'objet S par un instrument modulaire FLSC

```
;;;; instrument virtuel Schaefferien

;; amp: un controle
;; freq: un controle
;; cal: un entier
;; nfunc: une fonction [entier -> entier]
;; mass: une fonction [nombre -> controle]
;; col: une fonction [controle -> controle]
;; harm: une fonction [nombre -> controle]
;; phi: une fonction [entier -> nombre]
;; end: un nombre

(define instrsch (lambda (amp freq cal nfunc mass col harm phi end)
                   ;; definir le support temporel
                   (base [0 end]
                     ;; indices
                     (let* ((indexes (range 1 (nfunc cal)))
                            ;; phases
                            (phases (map phi indexes))
                            ;; frequences relatives
                            (lfrel (map mass indexes))
                            ;; frequences absolues
                            (lfabs (m* freq lfrel)))
                       ;; iterer sur les composantes
                       (map (lambda (ind fabs phase)
                              ;; appliquer la dynamique
                              (mn* amp
                                ;; appliquer l'enveloppe relative
                                (harm ind)
                                ;; appliquer l'enveloppe absolue
                                (col fabs)
                                ;; generer le signal
                                (osc fabs phase)))
                         ;; listes a traiter
                         indexes lfabs phases)))))
```

3 Typologie des fonctions Schaefferiennes

Nous allons maintenant examiner plus en détail les définitions possibles des fonctions que nous avons énoncées en 1. Nous procèderons à une analyse des problématiques spécifiques à chacune d'entre elles, et proposerons les éléments de solution potentiels que nous avons étudié. Bien entendu, de nombreuses autres solutions peuvent être envisagées, ce formalisme restant ouvert à une expressivité potentiellement illimitée.

La description des objets Schaefferiens se ramène essentiellement à deux grands types de critères : d'une part les critères de *forme* qui décrivent l'évolution de l'objet au cours du temps, à des fréquences que la perception humaine interprètera comme un parcours des configurations possibles, et d'autre part les critères de *matière*, qui ont trait à la composition spectrale de l'objet au cours d'une fenêtre temporelle qui pour la perception constitue un état instantané du phénomène sonore.

3.1 Forme

Le critère de forme regroupe l'ensemble des informations qui qualifient l'évolution de état de l'objet au cours du temps ; il s'agit donc d'évolutions suffisamment lentes des fonctions temporelles le caractérisant pour que la perception humaine puisse distinguer la nuance entre un instant et l'instant suivant. Il est difficile de donner un ordre de grandeur de la limite dans laquelle cette notion s'applique, mais on peut considérer comme un repère la fréquence sonore à partir de laquelle le phénomène sonore associé à une onde acoustique semble acquérir un caractère homogène au cours du temps.

Nous ramenons ce problème à la description des paramètres (fonctions temporelles de type contrôle) qui déterminent le processus de synthèse sonore. Il s'agit donc de décrire en premier lieu la *temporalité* de l'objet, c'est à dire la disposition de ses points d'inflexion dans le temps, puis la façon dont celle-ci est occupée par ses *paramètres*, qui sont construits au moyen de fonctions d'*enveloppe* et de *modulateurs*.

Les paramètres principaux d'un objet sont sa dynamique d^P (1.1) et son profil mélodique f_0^P (1.2), mais nous verrons en 3.2 que d'autres critères peuvent apparaître suivant la spécification du matériau sonore.

3.1.1 Temporalité

Afin de spécifier les critères de forme de l'objet, il est nécessaire de déterminer sa temporalité. Celle-ci est constituée par son *support temporel* $[T_0, T_L]$ et sa *signature temporelle* $(T_l)_L$ (cf. 1.3 page 23). On constate que le support peut se déduire de la signature, sur laquelle on va désormais se concentrer.

La signature pourrait être calculée à partir d'une simple séquence de durées, mais nous allons opter pour un modèle légèrement plus élaboré, qui permettra une spécification plus signifiante. Les durées qui composent ce modèle sont :

Temps longs : ils représentent les durées les plus longues (p.ex. $0,25s$ à $4s$ ou plus) employées et forment la structure globale de la signature ; ils peuvent être en nombre arbitraire (éventuellement 0 pour des objets très ponctuels

Temps courts : il s'agit de points-clef ajoutés à proximité des temps longs, et qui n'ont pas d'influence sur ceux-ci ; ce procédé permet d'ajouter des transitions rapides (p.ex. $1ms$ à $256ms$) à l'intérieur d'un objet sans modifier sa structure globale

Temps anticipés : ils sont analogues au temps courts, mais placés *avant* les points-clef (temps longs uniquement) ; leur fonction est la même, sauf qu'ils permettent d'ajouter des transitions rapides vers un point-clef plutôt qu'à la suite de celui-ci

Si on note `L` les temps longs, `C` les temps courts, et `A` les temps anticipés, la structure de la signature peut être représentée par l'expression régulière suivante :

```
C*(A*LC*)*
```

Cette séquence de durées $(\theta_l)_{L-1}$ est constituée d'éléments qui peuvent être exprimés de la façon suivante :

$\forall i \in \{0, \ldots, L-1\} : \theta_l \in \{(type, T) | type \in \{long, court, antcp\}, T \in \mathbb{T} > 0\}$

La signature $(T_l)_L$ peut alors être construite au moyen de la fonction récursive suivante (supposant un temps d'intervalle minimal T_Δ entre les points-clef) :

$Soit\,(T_\Delta > 0) \in \mathbb{T}$:
$signrec : (\theta_k)_K, T_{lng}, T_{crt}, T_{ac} \mapsto$

$$\begin{cases} si\,K = -1 : & \big((), \emptyset\big) \\ sinon : & soit\,(type, T) = \theta_0 \\ & soit\,(\theta'_k)_{K-1}\,t.q.\,\theta'_k = \theta_{k+1} \\ & \begin{cases} si\,type = long : & soit\,T_{cour} = max(T_{lng} + T, T_{lng} + T_{crt} + T_\Delta) \\ & soit\,((T_k)_{K-1}, T_{ref}) = signrec\big((\theta'_k)_{K-1}, T_{cour}, 0, 0\big) \\ & \big((T_{cour}, T_0, \ldots, T_{K-1})_K, T_{cour} - T_{ac}\big) \\ si\,type = court : & soit\,T_{cour} = T_{crt} + T \\ & soit\,((T_k)_{K-1}, T_{ref}) = signrec\big((\theta'_k)_{K-1}, T_{lng}, T_{cour}, 0\big) \\ & \big((T_{lng} + T_{cour}, T_0, \ldots, T_{K-1})_K, \emptyset\big) \\ si\,type = antcp : & soit\,T_{cour} = T_{ac} + T \\ & soit\,((T_k)_{K-1}, T_{ref}) = signrec\big((\theta'_k)_{K-1}, T_{lng}, T_{crt}, T_{cour}\big) \\ & si\,T_{ref} = \emptyset : erreur, sinon : \\ & \big((T_{ref} + T_{ac}, T_0, \ldots, T_{K-1})_K, T_{ref}\big) \end{cases} \end{cases}$$

La signature associée à $(\theta_l)_{L-1}$ sera alors :
$Soit\,\big((T'_l)_{L-1}, T_{ref}\big) = signrec\big((\theta_l)_{L-1}, 0, 0, 0\big)$
$Alors\,(T_l)_L = (0, T'_0, \ldots, T'_{L-1})_L$

3.1.2 Enveloppes

Les évolutions temporelles des paramètres peuvent être décrites au moyen de fonctions d'enveloppe (cf. Fonctions d'enveloppe) définies par une séquence de valeurs $(a_i)_I$ associées aux temps $(T'_i)_I$.

La temporalité de l'objet étant déjà décrite par sa signature $(T_l)_L$, et les valeurs d'enveloppe pouvant être normalisées sur $[-1, 1]$, on peut ramener la description d'une enveloppe à une séquence $(v_l)_L$ telle que $\forall l \in \{0, \ldots, L\} : v_l \in [-1, 1] \cup \{\emptyset\}$.

Alors, on peut définir I de la façon suivante :
$I = card(\{i \in \{0, \ldots, L\} | v_i \neq \emptyset\})$
Ensuite, $(k_i)_I$ est la séquence d'indices $k_i \in \{0, \ldots, L\}$ telle que :
$\forall i \in \{0, \ldots, I\} : v_{k_i} \neq \emptyset\, et\, \forall j \in \{0, \ldots, i-1\} : k_j < k_i$
Les $(T'_i)_I$ et $(a_i)_I$ sont finalement définis comme :
$\forall i \in \{0, \ldots, I\} : T'_i = T_{k_i}$
$\forall i \in \{0, \ldots, I\} : a_i = v_{k_i}$
L'enveloppe souhaitée est alors l'objet $sign_{(T'_i)_I}\big(env^O(a_0, \ldots, a_i)\big)$; nous le noterons ici $penv\big((v_l)_L\big)$.

3.1.3 Modulateurs

Le critère rythmique correspond aux notions Schaefferiennes de *grain* et d'*allure* ; chacune de ces deux notions décrit des modulations périodiques ou pseudo-periodiques de l'énergie et/ou de la hauteur de l'objet, qui sont habituellement synchrones. Elles diffèrent essentiellement par la bande de fréquence des modulations engendrées.

Nous effectuons ici une généralisation de ces notions en considérant que ces modulations peuvent s'appliquer à n'importe lequel des paramètres qui décrivent l'objet, et pas seulement aux profil dynamique d^P et au profil mélodique f_0^P.

Etant donné qu'il est possible de les normaliser sur $[-1, 1]$, elles peuvent être décrites simplement au moyen des fonctions pseudo-périodiques décrites dans la sous-section "Fonctions pseudo-périodiques" comme l'objet $mod_x^O(f, \phi)$. Nous les noterons ici $pmod(x, f, \phi)$.

3.1.4 Paramètres

Les paramètres décrivant un objet, qu'il s'agisse de d^P, f_0^P ou des fonctions temporelles passées en paramètre aux transformations de matière (voir 3.2 page suivante), peuvent finalement être définis au moyen de constantes, d'enveloppes (cf. 3.1.2) et de modulateurs (cf. 3.1.3) normalisés, et de la temporalité de l'objet (cf. 3.1.1).

Nous allons tout d'abord présenter deux fonctions utilitaires employées ici :

Gain : afin de passer des fonctions définies sur $[-1,1]$ à des coefficients de gain sur $[0,1]$, nous utiliserons la fonction $gain : x \in [-1,1]^P \mapsto (t \mapsto 2 \cdot x(t) - 1)$

Profondeur : afin de pouvoir appliquer certains paramètres avec une pondération, nous utiliserons la fonction $prof : a \in [0,1]^P, x \in [0,1]^P \mapsto \big(t \mapsto 1 - a(t) + a(t) \cdot x(t) = 1 - a(t) \cdot (1 - x(t))\big)$

Les paramètres sont alors de deux types distincts :

Paramètres linéaires : ceux-ci décrivent des grandeurs dont la variation est linéaire, comme par exemple les gains, mais également un bon nombre d'indices de transformation divers ; on les regroupe dans l'ensemble $\mathcal{P}_{lin}$

Paramètres exponentiels : ceux-ci décrivent des grandeurs dont la variation est exponentielle, le plus souvent des fréquences (audibles ou non) ; on les regroupe dans l'ensemble $\mathcal{P}_{exp}$

Paramètres linéaires :

Un paramètre linéaire $plin\big(K, e, (p_i, m_i)_I\big) \in \mathcal{P}_{lin}$ est défini par une borne constante $K \in \mathbb{R}^+$, ainsi que par une enveloppe ou une constante $e = penv\big((v_l)_L\big)\ out\ t \mapsto K_e \in [-1,1]$ et une séquence de couples (profondeur, modulateur) $(p_i, m_i)_I$ telle que :

$p_i \in \big\{plin\big(1, e'', (p''_k, m''_k)\big) \in \mathcal{P}_{lin}\big\}$,

$$m_i \in \begin{Bmatrix} pmod(x, f, \phi)| \\ x \in \{sin, squ, tri, saw, \ldots\}, \\ f \in \big\{pexp\big(8, 5, e', (p'_j, m'_j)\big) \in \mathcal{P}_{exp}\big\}, \\ \phi \in]-\pi, \pi] \end{Bmatrix}$$

La fonction temporelle engendrée par un tel paramètre est la suivante :

$$plin : K, e, (m_i)_I \mapsto \Big(t \mapsto K \cdot gain(e^R)(t) \cdot \prod_{j=0}^{I} prof\big(p_j, gain(m_j^R)\big)(t)\Big)$$

Paramètres exponentiels :

Un paramètre exponentiel $pexp\big(K_{ref}, K_{mod}, e, (p_i, m_i)_I\big) \in \mathcal{P}_{exp}$ est défini par une valeur de référence constante $K_{ref} \in \mathbb{R}^+$, un indice de modulation constant $K_{mod} \in \mathbb{R}^+$, ainsi que par une enveloppe ou une constante $e = penv\big((v_l)_L\big)\ out\ t \mapsto K_e \in [-1,1]$ et une séquence de couples (amplitude, modulateur) $(p_i, m_i)_I$ telle que :

$p_i \in \big\{plin\big(1, e'', (p''_k, m''_k)\big) \in \mathcal{P}_{lin}\big\}$,

$$m_i \in \begin{Bmatrix} pmod(x, f, \phi)| \\ x \in \{sin, squ, tri, saw, \ldots\}, \\ f \in \big\{pexp\big(8, 5, e', (p'_j, m'_j)\big) \in \mathcal{P}_{exp}\big\}, \\ \phi \in]-\pi, \pi] \end{Bmatrix}$$

La fonction temporelle engendrée par un tel paramètre est la suivante :

$$pexp : K_{ref}, K_{mod}, e, (p_i, m_i)_I \mapsto \left(t \mapsto K_{ref} \cdot 2^{K_{mod} \cdot \left(e^R(t) + \sum_{j=0}^{I} p_j(t) \cdot m_j^R(t)\right)}\right)$$

Si on souhaite que cette fonction reste à l'intérieur de l'intervalle $[K_{ref} \cdot 2^{-K_{mod}}, K_{ref} \cdot 2^{K_{mod}}]$, il est nécessaire de faire en sorte que pour tout $t \in [T_0, T_L]$, la somme des fonctions modulantes $e^R(t) + \sum_{j=0}^{I} p_j(t) \cdot m_j^R(t)$ soit comprise dans $[-1,1]$. Cela peut se faire en affectant les niveaux de e ainsi que la constante ou les niveaux d'enveloppe des p_i d'un ensemble de coefficients dont la somme est 1.

3.2 Matière

Le critère de matière est le complémentaire du critère de forme, et représente ce qui, du point de vue de la perception humaine, constitue un état instantané du phénomène sonore, qui ne peut être décomposé plus avant. Il s'agit donc pour nous des fonctions qui caractérisent, en termes de fréquences et d'amplitudes, la composition spectrale du signal construit au cours d'une fenêtre temporelle suffisamment courte pour être interprétée comme un tout.

La théorie Schaefferienne décompose la matière en *masse* (le type de sonorité dont il est question) et en *timbre harmonique* (la situation précise du phénomène à l'intérieur de l'ensemble des phénomènes de même masse). Nous avons choisi d'associer cette différenciation à celle entre les altérations des *fréquences* des composantes (qui déterminent par exemple si un son est harmonique ou non, et si il est ou non un bruit) et les altérations

de l'*amplitude* des composantes (qui détermineront par exemple si un son est brillant ou mat, si il est riche ou sinon clair ou voilé, etc...).

Il faut bien prendre en compte que la plupart de ces altérations (sinon toutes) dépendent de paramètres qui évoluent potentiellement au cours du temps, engendrant autant d'opportunités supplémentaires de moduler la forme (cf. 3.1) de l'objet.

Les fonctions qui définissent ces critères sont caractérisées par des expressions algébriques, mais on doit se souvenir (cf. 1.4 et 3.1.5) que les opérations algébriques peuvent également s'appliquer à des fonctions temporelles ou des objets temporels. En l'occurence les paramètres de ces fonctions sont données par des objets temporels, suivant les critères de forme donnés ci-dessus.

3.2.1 Masse

Les modifications de la masse permettent de calculer le tuple (N, M, M', B, C) des fonctions qui la définissent :

- N est la fonction de remplissage (cf. 1.3 page 59)
- M est la fonction de masse (cf. 1.4 page 59)
- M' est la fonction de masse auxiliaire, B est la fonction de bruitage, C est la fonction de calibre dynamique ; celles-ci seront employées dans le calcul du timbre harmonique (voir 3.2.2 page 69)

On obtient ce résultat par la composition d'un sous-ensemble des fonctions particulières décrites ci-après :

Calibre : le calibre est la bande spectrale occupée par la masse ; il est possible de moduler cette grandeur au cours du temps

Redoublement : le redoublement consiste à diviser chaque composante en deux composantes distinctes, en conservant la fréquence moyenne

Bruitage : le bruitage consiste à moduler stochastiquement la fréquence des composantes, pour obtenir des bandes de bruit

Surdensité : la surdensité consiste à introduire des composantes intermédiaires entre les composantes d'origine

Distorsion quadratique : la distorsion quadratique consiste à élever les fréquences des composantes à une puissance comprise entre 0 et 1

Distrorsion exponentielle : la distorsion exponentielle consiste à multiplier la fréquence des composantes par un facteur exponentiel de leur indice

Décalage : le décalage consiste à soustraire un même terme à la fréquence des composantes

Ces transformations, si elles sont effectuées, doivent l'être dans un ordre prédéfini, de façon à maintenir la cohérence des expressions. Nous allons donc les présenter dans l'ordre dans lequel elles peuvent être effectuées. Nous terminerons en donnant l'expression qui permet d'effectuer l'assemblage des fonctions sélectionnées.

Redoublement :

Le redoublement consiste à diviser chaque composante en deux composantes distinctes, en conservant la fréquence moyenne. Dans ce but, le nombre de composantes est multiplié par 2, puis la fréquence relative des composantes impaires est diminuée de $\delta \in [0, 1]$, et celle des composantes paires est augmentée de δ.

Ceci permet d'obtenir des sons affectés d'une légère inharmonicité, à la manière des cloches ; les cloches n'étant jamais parfaitement circulaires, elles produisent deux hauteurs distinctes [1].

Cette transformation est exprimée de la façon suivante :

$red : (\delta \in \mathbb{T} \to [0, 1])$

$$\mapsto \left(\begin{array}{cl} & \Big(n \mapsto 2 \cdot N(n), \\ & i \mapsto \Big(t \mapsto M\big(ceil(\frac{i}{2})\big)(t) + \frac{\delta(t)}{2} \cdot (-1)^{i[2]}\Big), \\ (N, M, M', B, C) \mapsto & i \mapsto \Big(t \mapsto M'\big(ceil(\frac{i}{2})\big)(t)\Big), \\ & i \mapsto \Big(t \mapsto B(i)(t)\Big), \\ & i \mapsto \Big(t \mapsto C(i)(t)\Big)\Big) \end{array} \right)$$

1. Dans ce but il vaudrait multiplier les fréquences par $2^{\pm\delta}$; mais ce résultat peut être obtenu par superposition de 2 sons harmoniques.

Bruitage :

Le bruitage des composantes peut s'obtenir par modulation stochastique de leur fréquence et de leur amplitude, comme l'indique Pierre Hanna dans son modèle CNSS [14]. Pour la modulation stochastique d'amplitude, voir 3.2.2.

Les indices de composantes étant à la base du calcul de leur fréquence, on peut obtenir ce résultat en effectuant un bruitage des indices, c'est-à-dire une modulation stochastique de leur valeur.

On définit alors le bruitage B comme une fonction temporelle des indices $i \in \{1, \ldots, N(n)\}$, à partir de l'indice de bruitage de masse $\delta : \mathbb{T} \to [0,1]$:

▶ $B(i) = i + \frac{\delta(t)}{2} \cdot Rand_{br,i}|_{-1}^{1}(t)$

Le bruitage est inopérant au temps t lorsque $\delta(t) = 0$.

$bruit : (\delta \in \mathbb{T} \to [0,1])$

$$\mapsto \left(\begin{array}{ll} & \Big(n \mapsto N(n), \\ & i \mapsto \Big(t \mapsto M(i)(t) + \dfrac{\delta(t)}{2} \cdot Rand_{br,i}(t)|_{-1}^{1} \Big), \\ (N, M, M', B, C) \mapsto & i \mapsto \Big(t \mapsto M'(i)(t) \Big), \\ & i \mapsto \Big(t \mapsto B(i)(t) + \dfrac{1}{2} \cdot Rand_{br,i}(t)|_{-1}^{1} \Big), \\ & i \mapsto \Big(t \mapsto C(i)(t) \Big) \Big) \end{array} \right)$$

Surdensité :

La surdensité consiste à introduire des composantes intermédiaires entre les composantes d'origine ; ceci peut être effectué S_{max} fois de suite, ce qui subdivise chaque intervalle en $2^{S_{max}}$. Le paramètre $S : \mathbb{T} \to [0, S_{max}]$ permet de moduler progressivement le nombre de composantes actives à l'instant t, en effectuant un fondu sur le niveau de subdivision le plus fin à cet instant.

Cette transformation permet d'obtenir deux effets distincts :

- pour les sons toniques, elle permet d'obtenir un effet de fondamentale absente
- pour les bruits, elle permet de renforcer la densité spectrale tout en conservant le calibre

Elle est exprimée de la façon suivante :

$surd : (S_{max} \in \mathbb{N}, S \in \mathbb{T} \to [0, S_{max}])$

$$\mapsto \left(\begin{array}{ll} & soit\, o : i \mapsto ceil(log_2(i[2^{S_{max}}] + 1)) \\ & soit\, \Delta : i \mapsto (2 \cdot i[2^{S_{max}}] + 1)[2^{o(i)}] \cdot 2^{-o(i)} \\ & \Big(n \mapsto \big(N(n) - 1 \big) \cdot 2^{S_{max}} + 1, \\ & i \mapsto \Big(t \mapsto M\big(ceil(\dfrac{i}{2^{S_{max}}}) \big)(t) + \Delta(i) \Big), \\ (N, M, M', B, C) \mapsto & i \mapsto \Big(t \mapsto M'\big(ceil(\dfrac{i}{2^{S_{max}}}) \big)(t) + \Delta(i) \Big), \\ & i \mapsto \Big(t \mapsto B(i)(t) \Big), \\ & i \mapsto \Big(t \mapsto C(i)(t) \cdot max\Big(0, min\big(1, S(t) - o(i) \big) \Big) \Big) \Big) \end{array} \right)$$

Calibre :

Il est possible de moduler le nombre de composantes audibles au cours du temps, en fixant dynamiquement par le paramètre x le gain des composantes, suivant leur fréquence relative $M'(i)(t)$.

Ceci peut être effecuté par la fonction suivante :

$cal : (x \in \mathbb{T} \to \mathbb{R}^+)$

$$\mapsto \left(\begin{array}{c} \Big(n \mapsto N(n), \\ i \mapsto \Big(t \mapsto M(i)(t) \Big), \\ (N, M, M', B, C) \mapsto i \mapsto \Big(t \mapsto M'(i)(t) \Big), \\ i \mapsto \Big(t \mapsto B(i)(t) \Big), \\ i \mapsto \Big(t \mapsto C(i)(t) \cdot \big(x(t) - M'(i)(t) \big)|_0^1 \Big) \Big) \end{array} \right)$$

Distorsion quadratique :

La distorsion quadratique consiste à élever les fréquences relatives des composantes à une puissance $\frac{1}{\beta}$comprise entre 0 et 1. Ce type de distorsion est fréquemment rencontré dans les sons produits par des membranes, en raison de la répartition de leurs modes propres ; dans ce cas $\beta = 2$. Il est possible de choisir d'autres valeurs de β, voire même de le varier au cours du temps, mais la correspondance avec les phénomènes psychoacoustiques est inconnue dans ce cas.

Il est possible d'effectuer un fondu des composantes de fréquence relative non-entière au moyen du paramètre $\delta : \mathbb{T} \to \mathbb{R}^+$; les composantes disparaissent au fur et à mesure que δ augmente.

Cette transformation est exprimée ainsi :

$distquad : (\beta \in \mathbb{T} \to [1, +\infty[, \delta \in \mathbb{T} \to \mathbb{R}^+)$

$$\mapsto \left(\begin{array}{c} \Big(n \mapsto N(n), \\ i \mapsto \Big(t \mapsto M(i)(t)^{\frac{1}{\beta(t)}} \Big), \\ (N, M, M', B, C) \mapsto i \mapsto \Big(t \mapsto M'(i)(t)^{\frac{1}{\beta(t)}} \Big), \\ i \mapsto \Big(t \mapsto B(i)(t) \Big), \\ i \mapsto \Big(t \mapsto C(i)(t) \cdot prox\big(\delta(t), M'(i)^{\frac{1}{\beta(t)}}\big) \Big) \Big) \end{array} \right)$$

Distorsion exponentielle :

La distorsion exponentielle consiste à multiplier la fréquence des composantes par un facteur exponentiel de leur indice multiplié par un paramètre $\beta : \mathbb{T} \to \mathbb{R}^+$. Lorsque $\beta(t) = 0$, la transformation est sans effet.

Ce type de phénomène se retrouve dans les sons produits par les cordes de piano, comme l'indique Bernard Cohen [8] ; ceci semble être dû à la non homogénéité du matériau, il est donc concevable que le phénomène se transporte à d'autres corps sonores linéiques et non-homogènes.

On l'exprime ainsi :

$distexp : (\beta \in \mathbb{T} \to \mathbb{R}^+)$

$$\mapsto \left(\begin{array}{c} \Big(n \mapsto N(n), \\ i \mapsto \Big(t \mapsto M(i)(t) \cdot 2^{\beta(t) \cdot M(t)(i)} \Big), \\ (N, M, M', B, C) \mapsto i \mapsto \Big(t \mapsto M'(i)(t) \Big), \\ i \mapsto \Big(t \mapsto B(i)(t) \Big), \\ i \mapsto \Big(t \mapsto C(i)(t) \Big) \Big) \end{array} \right)$$

Décalage :

Cette transformation consiste à soustraire un même terme $\delta : \mathbb{T} \to [0, 1]$ à la fréquence des composantes, ce qui produit un décalage inharmonique (*frequency shift*). Cet effet relativement utilisé ne semble correspondre à

aucun phénomène naturel, néanmoins il est inclus pour des raisons de complétude : la transformation affine est la seule transformation linéaire simple qui n'est pas couverte par les autres fonctions.

Elle s'exprime comme suit :

$decl : (\delta \in \mathbb{T} \to [0,1])$

$$\mapsto \begin{pmatrix} \Big(n \mapsto N(n), \\ i \mapsto \Big(t \mapsto M(i)(t) - \delta(t)\Big), \\ (N,M,M',B,C) \mapsto i \mapsto \Big(t \mapsto M'(i)(t)\Big), \\ i \mapsto \Big(t \mapsto B(i)(t)\Big), \\ i \mapsto \Big(t \mapsto C(i)(t)\Big)\Big) \end{pmatrix}$$

Application :

Les fonctions ci-dessus peuvent individuellement être appliquées ou non à une masse donnée, il est donc nécessaire de pouvoir sélectionner celles qui sont souhaitées et de les appliquer dans le bon ordre. Nous allons donc présenter les outils mathématiques qui permettent de réaliser cette opération.

La fonction récursive *masserec* prend en argument une séquence de couples fonction-paramètres, ainsi que le tuple de fonctions (N, M, M', B, C) d'origine.

Cette fonction est exprimée ainsi :

$masserec : \big((f_1,p_1),\dots,(f_k,p_k)\big),(N,M,M',B,C)$

$$\mapsto \begin{cases} si\ k=0: & (N,M,M',B,C) \\ si\ k>0: & soit\ f = f_k(p_k) \\ & dans\ masserec\Big(\big((f_1,p_1),\dots,(f_{k-1},p_{k-1})\big), f'(N,M,M',B,C)\Big) \end{cases}$$

Les couples (f,p) correspondent aux transformations effectuées, ils constituent l'ensemble $\mathcal{E}_M$ dans lequel on a ajouté l'information de l'ordre d'effection :

$$\begin{aligned} \mathcal{E}_M = &\{(1,(cal,(x)))|x \in \mathbb{T} \to \mathbb{R}^+\} \\ &\cup \{(2,(red,(\delta)))|\delta \in \mathbb{T} \to [0,1]\} \\ &\cup \{(3,(bruit,(\delta)))|\delta \in \mathbb{T} \to [0,1]\} \\ &\cup \{(4,(surd,(S_{max},S)))|S_{max} \in \mathbb{N}, S \in \mathbb{T} \to [0,S_{max}]\} \\ &\cup \{(5,(distquad,(\beta,\delta)))|\beta \in \mathbb{T} \to [1,+\infty[, \delta \in \mathbb{T} \to \mathbb{R}^+\} \\ &\cup \{(6,(distexp,(\beta)))|\beta \in \mathbb{T} \to \mathbb{R}^+\} \\ &\cup \{(7,(decl,(\delta)))|\delta \in \mathbb{T} \to [0,1]\} \end{aligned}$$

A partir d'un ensemble $E_M \subset \mathcal{E}_M$ contenant les transformations souhaitées (sans répétition parmi un type de transformation), il est possible d'obtenir une séquence ordonné (m_k) qui pourra être passée en argument à *masserec*.

On peut à partir de là dégager l'expression de la fonction *masse*, qui à un ensmeble E_M associe le tuple souhaité :

$Soit\ E_M \subset \mathcal{E}_M\ t.q. \forall a_1 = (n_1,f_1,p_1), a_2 = (n_2,f_2,p_2) \in E_M : a_1 \neq a_2 \implies n_1 \neq n_2$

$Soient\ (m'_k)_{card(E)}, (m_k)_{card(E)}\ t.q. \forall k, (m'_k, m_k) \in E_M\ et \forall j < k : m'_j > m'_k$

$$Alors\ masse(E_M) = (N,M,M',B,C) = masserec\Big((m_k), \begin{pmatrix} (n \mapsto n \\ t \mapsto (i \mapsto i), \\ (1,1), t \mapsto (i \mapsto i), \\ t \mapsto (i \mapsto i), \\ t \mapsto (i \mapsto 0), \\ t \mapsto (i \mapsto 1)) \end{pmatrix}\Big)$$

3.2.2 Timbre harmonique

La fonction de timbre harmonique H est calculée suivant les fonctions d'enveloppe spectrale relative et absolue, respectivement H_R et H_A, ainsi que des fonctions de bruitage B et de calibre dynamique C. La fonction H_R dépend de M', et la fonction H_A dépend de f, et donc de f_0 et M.

La fonction H est *in fine* une fonction du type $\mathbb{N}^* \to (\mathbb{T} \to \mathbb{R}^+)$ qui à l'indice i associe la fonction temporelle du gain de la $i^{ème}$ composante au temps t.

Ces deux fonctions peuvent être calculées par le produit d'un ensemble de fonctions choisies parmi les primitives suivantes :

Pour H_R :

Bruitage harmonique : ce procédé consiste à moduler l'amplitude des composantes de façon stochastique, pour obtenir un bruit

Filtre en peigne : on ramène l'allure spectrale vers celle d'un signal harmonique en filtrant suivant la différence des fréquences des composantes avec un multiple de la fréquence de base

Dégressivité : on affecte les composantes d'un gain plus ou moins décroissant suivant l'indice de partiel

Dominance : on affecte les premières composantes d'un même gain, produisant une marche dans l'allure spectrale à une fréquence relative déterminée

Présence cyclique : on affecte les composantes d'un gain qui dépend de leur place dans un cycle entier, calculée par un modulo

Présence logarithmique : on affecte les composantes d'un gain qui dépend de leur proximité à un ensemble de puissances de 2 de nombre impairs, calculée par des logarithmes

A cela vient s'ajouter la fonction de calibre dynamique C calculée lors de la construction des fonctions de masse, ainsi que la fonction B_{mod} qui permet de moduler les bandes de bruit suivant la fonction B calculée ci-dessus.

Pour H_A, il n'existe à l'heure actuelle qu'une seule primitive : la *couleur*, qui permet par combinaison de construire un égaliseur arbitraire.

Nous allons présenter dans le détail ces différentes fonctions, avant d'expliquer comment elles peuvent être combinées pour obtenir la fonction H.

Calibre dynamique :

Il est possible de moduler dynamiquement le calibre au moyen de la fonction C qui associe au cours du temps un gain à l'indice de la composante, régulant ainsi sa présence. Celle-ci est construite au cours de l'évaluation des fonctions de masse (cf. 3.2.1 page 66).

Le calibre dynamique est défini comme :

■ $C : \mathbb{N}^* \to (\mathbb{T} \to \mathbb{R}^+)$: une fonctionnelle temporelle

Celui-ci peut être neutre si $\forall t, i : C(i)(t) = 1$. Le calibre dynamique est lié à certains paramètres, suivant la composition de la transformation de masse ; certaines valeurs de paramètres peuvent neutraliser certains éléments de cette transformation.

Modulation du bruit :

Il est possible de moduler l'allure des bandes de bruit, en appliquant un gain calculé par l'application de la fonction *prox* à la mesure de bruit B. L'impact de la modification d'allure peut être réglé par le paramètre δ : lorsque $\delta(t) = 0$, il n'y a aucune modulation.

On peut préférer se passer de cette modulation, elle est donc définie de façon contitionnelle :

■ $B_{mod} : \begin{cases} (\delta \in \mathbb{T} \to \mathbb{R}^+) \mapsto \Big(i \mapsto \big(t \mapsto prox\big(\delta(t), B(t)(i)\big)\big)\Big) \\ OU \quad i \mapsto (t \mapsto 1) \end{cases}$

Bruitage harmonique :

Le timbre harmonique peut être affecté d'un bruitage (modulation stochastique de l'amplitude au cours du temps). On doit ici utiliser *CRand* afin d'éviter les discontinuités du signal résultant qui introduiraient des artefacts. L'impact sur l'amplitude des composantes est calculé par la fonction suivante, à partir de l'indice de bruitage timbral $\alpha : \mathbb{T} \to [0, 1]$:

■ $bharm : (\alpha \in \mathbb{T} \to [0, 1]) \mapsto \big(x \mapsto (t \mapsto 1 - \alpha \cdot CRand_{bh,x}|_0^1(t))\big)$

Le cas bruité est équivalent au cas non-bruité lorsque $\alpha = 0$.

Filtre en peigne :

Le filtre en peigne est le format de filtre le plus simple possible : il affecte les composantes d'un gain suivant la proximité de leur fréquence relative avec un nombre entier. Cela signifie que le signal résultant se rapproche d'un signal harmonique, lorsque l'intensité de l'effet augmente. Celle-ci peut être réglée au moyen du paramètre δ ; l'effet est neutralisé lorsque $\delta(t) = 0$.

■ $comb : (\delta \in \mathbb{T} \to \mathbb{R}^+) \mapsto \Big(x \mapsto \big(t \mapsto prox\big(\delta(t), x\big)\big)\Big)$

Dégressivité :

L'intensité des composantes, dans la réalité, est le plus souvent décroissante suivant leur fréquence relative. Les sons dont les composantes aigües sont puissantes sont qualifiés de *brillants*, lorsque c'est l'inverse on les qualifie de *mats*. Il est fortement conseillé d'employer la fonction suivante, qui atténue les composantes suivant leur fréquence relative et le paramètre s, avec une pente d'environ $-3 \cdot s\, dB/oct$, sachant que $-3\, dB/oct$ est un minimum raisonnable dans la plupart des cas :

■ $degr : (s \in \mathbb{T} \to \mathbb{R}^+) \mapsto \big(x \mapsto (t \mapsto x^{-s(t)})\big)$

Dominance :

La dominance permet de renforcer la présence des premières composantes jusqu'à la fréquence relative $x_0(t)$, en les affectant d'un gain $\alpha(t)$. Selon nos études, il s'agit d'un élément fondamental qui contribue à engendrer un son que l'on peut qualifier de *clair*.

■ $dom : (\alpha \in \mathbb{T} \to \mathbb{R}^+, x_0 \in \mathbb{T} \to \mathbb{R}^+) \mapsto \big(x \mapsto (t \mapsto \alpha^{(x - x_0(t))|_0^1})\big)$

Présence cyclique :

La présence cyclique consiste à affecter les composantes d'un gain cyclique suivant la fréquence relative, au travers d'une séquence de gains $(\alpha_i(t))_I$ qui se répète donc toutes les I fréquences relatives entières. Lorsque $I = 2$, ceci peut produire une enveloppe spectrale caractéristique des instruments bois tels que le hautbois ou le cor anglais, ou encore les oscillateurs triangulaires ou carrés. D'autres valeurs de I peuvent éventuellement produire d'autres résultats non étudiés.

Cette fonction s'exprime comme suit :

■ $p_{cyc} : \big(I \in \mathbb{N}, (\alpha_i)_I, \alpha_i \in \mathbb{T} \to [0,1]\big) \mapsto \Big(x \mapsto \big(t \mapsto \sum_{i=1}^{I} \alpha_i(t) \cdot prox(I, \frac{x-i}{I})\big)\Big)$

Présence logarithmique :

La présence logarithmique consiste à filtrer les composantes suivant la proximité de leur fréquence relative à une puissance de 2 multipliée par l'un des I premiers nombres impairs (dont 1). On obtient un son caractéristique des premiers sons d'orgue électronique, avec des composantes à la quinte, la tierce ou la septième lorsque $I > 1$. Ces sons peuvent être qualifiés de *groupes* car les composantes restantes entretiennent une indépendance relative.

La fonction associée est la suivante :

■ $p_{log} : \big(I \in \mathbb{N}, (\alpha_i)_I, \alpha_i \in \mathbb{T} \to [0,1], \delta \in \mathbb{T} \to \mathbb{R}^+\big) \mapsto \Big(x \mapsto \big(t \mapsto \max_{i=1}^{I} \alpha_i(t) \cdot prox\big(\delta(t), log_2(\frac{x}{2 \cdot i + 1})\big)\big)\Big)$

Couleur :

Le filtrage suivant la fréquence absolue revient à effectuer l'opération habituellement connue sous le nom d'*égalisation*. Cette opération est tellement courante dans l'ingénierie du son qu'il est inutile d'énumérer la variété de résultats qu'elle permet d'atteindre.

Puisque le modèle de synthèse est modulaire, on peut produire ici ce type de résultat en cumulant des filtres élémentaires qui appliquent progressivement un gain de $\pm 3 \cdot \alpha(t)$ entre les fréquences $f_0(t)$ et $f_1(t)$. Lorsque ces deux fréquences sont inversées, le gain est appliqué dans l'autre sens. En dehors de ces bornes, le gain est constant.

Ceci est exprimé par la fonction suivante :

■ $col : (\alpha \in \mathbb{T} \to \mathbb{R}, f_0 \in \mathbb{T} \to \mathbb{R}^+, f_1 \in \mathbb{T} \to \mathbb{R}^+) \mapsto \big(f \mapsto (t \mapsto 2^{\alpha(t)^{(\frac{2 \cdot f - (f_1(t) + f_0(t))}{(f_1(t) - f_0(t))})|_{-1}^1}})\big)$

Synthèse :

Le timbre harmonique H résulte de la combinaison des fonctions sélectionnées, ainsi que des fonctions B et C. Cette combinaison peut être définie comme un sous ensemble des ensembles $\mathcal{E}_H$ et $\mathcal{E}'_H$, respectivement pour les fonctions relatives et absolues :

- $\mathcal{E}_H = \Big\{ \big(comb, (\delta)\big) | \delta \in \mathbb{T} \to \mathbb{R}^+ \Big\}$
  $\cup \Big\{ \big(bharm, (\alpha)\big) | \alpha \in \mathbb{T} \to [0,1] \Big\}$
  $\cup \Big\{ \big(degr, (s)\big) | s \in \mathbb{T} \to \mathbb{R}^+ \Big\}$
  $\cup \Big\{ \big(dom, (\alpha, x_0)\big) | \alpha \in \mathbb{T} \to \mathbb{R}^+, x_0 \in \mathbb{T} \to \mathbb{R}^+ \Big\}$
  $\cup \Big\{ \Big(p_{cyc}, \big(I, (\alpha_i)_I\big)\Big) | I \in \mathbb{N}, \alpha_i \in \mathbb{T} \to [0,1] \Big\}$
  $\cup \Big\{ \Big(p_{log}, \big(I, (\alpha_i)_I, \delta\big)\Big) | I \in \mathbb{N}, \alpha_i \in \mathbb{T} \to [0,1], \delta \in \mathbb{T} \to \mathbb{R}^+ \Big\}$
- $\mathcal{E}'_H = \Big\{ \big(col, (\alpha, f_0, f_1)\big) | \alpha \in \mathbb{T} \to \mathbb{R}, f_0 \in \mathbb{T} \to \mathbb{R}^+, f_1 \in \mathbb{T} \to \mathbb{R}^+ \Big\}$
- $E_H \subset \mathcal{E}_H$, $E'_H \subset \mathcal{E}'_H$

La combinaison des fonctions se fait ensuite au moyen d'une fonction produit H_{prod}, qui sera appliquée à ces deux ensembles :

- $H_{prod} : E \mapsto \left(x \mapsto \left(t \mapsto \prod_{(f,p) \in E} f(p)(x)(t) \right) \right)$

On peut alors obtenir H_R, H_A et H de la façon suivante :

- $H_R : i \mapsto \Big(t \mapsto H_{prod}(E_H)\big(M'(i)\big)(t) \cdot C(i)(t) \cdot B_{mod}(i)(t) \Big)$
- $H_A = H_{prod}(E'_H)$
- $H : i \mapsto \Big(t \mapsto H_A\big(f_A(i)(t)\big)(t) \cdot H_R(i)(t) \Big)$

4 Conclusion

Nous avons formulé un système complet pour la synthèse sonore, qui permet d'exprimer les notions de profil mélodique, dynamique, harmonique et de masse, les décomposant en critères de forme et critères de matière ; ceci rejoint les distinctions typologiques élaborées par Pierre Schaeffer. Au passage, cela illustre comment les critères de matière donnent potentiellement lieu à de nombreuses catégories ; en effet, même si le champ spectral constitue un continuum, il n'en est pas de même pour l'ensemble des fonctions analytiques nécessaires pour le décrire ; ceci nous montre en quoi l'impératif d'invariance algorithmique (cf. 5.1.1 page 32) constitue une contrainte forte sur la programmation musicale.

Ce système nous permettra de proposer un outil pour la synthèse sonore, qui sera à la fois expressif et intuitif. Expressif, car au travers des nombreux modules permettant d'obtenir des timbres et des masses différentes, ainsi que du fait que tout les paramètres peuvent évoluer de façon complexe au cours du temps, il offre à l'utilisateur une palette d'expression extrêmement large ; intuitif, car il mettra à la portée de personnes n'ayant que peu d'expérience en informatique le potentiel d'un langage de programmation complet, au sein duquel ils pourront s'orienter simplement par une écoute attentive, les éléments du système étant conçus pour s'adresser à la perception humaine.

Chapitre V

Application à la musicologie et à la psychoacoustique

Nous avons pu formuler un certain nombre de schémas de synthèse sonore, dont nous espérons qu'ils sont pertinents dans l'audition humaine. Il s'agit ici de mesurer, dans la mesure du possible, la valeur de ces hypothèses ; c'est à dire de donner à entendre les sons associés à des auditeurs humains, afin d'évaluer leur reconnaissance ou leur non reconnaissance des critères modélisés.

Nous allons donc faire état d'un ensemble d'expériences que nous avons entreprises dans le cadre de ce travail, et de la recherche concomitante d'une méthodologie cohérente (notamment d'après les connaissances en psychoacoustique) et adéquate (c'est à dire compatible avec les impératifs pratiques). Les trois expériences présentées ci-après correspondent à trois étapes d'évolution de notre façon d'envisager la recherche musicale, tout d'abord d'une façon classique (1), puis par les sciences expérimentales (2), et finalement en s'orientant en direction des sciences sociales 3).

1 Modélisation de critères musicologiques

Nous décrivons ici la méthode par laquelle nous avons modélisé, en termes de synthèse sonore, un critère musicologique se situant dans la théorie de Jean-Louis Di Santo, complémentaire par la dualité forme-matière de sa notion d'*unités minimales* [11].

Nous exposons donc les principes fondamentaux sous-jacents à cette expérience, la situation de ce critère au sein de la théorie, la méthodologie employée, les résultats obtenus, ainsi que les développements possibles de cette démarche.

Cette expérience s'est étalée sur de nombreux mois, au cours de l'année 2017, et a nécessité un travail et un engagement considérable de la part de Jean-Louis Di Santo et nous-mêmes. A posteriori il semble évident que la reconstruction d'une théorie entière, et non d'un seul critère, constitue une tâche difficile qui requiert des moyens humains conséquents ; l'enjeu d'un tel travail étant bien entendu une source de progrès importante pour la musique électroacoustique, sans compter d'éventuelles conséquences en psychoacoustique.

1.1 Principe de l'expérience

Cette expérience consiste essentiellement à établir la relation entre un critère musicologique identifiable par la perception seule, et l'expression mathématique de signaux acoustiques canoniques dont on peut prédire si ils évoqueront ou non ce phénomène de la perception.

Ceci suppose de s'appuyer sur une description formelle des objets audionumériques en question, grâce à laquelle il est possible de discerner l'appartenance catégorielle desdits objets dans la perception humaine.

1.1.1 Le modèle formel comme repère objectif

Le modèle FLSC permet une expression mathématiquement exacte des signaux sonores qui servent de support à la perception ; il faut bien comprendre ici que les phénomènes acoustiques mis en jeu varient quant à eux suivant les systèmes de rendu et les conditions d'écoute. Cette objectivité ne peut donc émerger dans la perception seulement par une reproduction répétée de l'expérience, ainsi qu'une compétence à l'écoute dont le développement requiert de nombreuses années d'étude.

Les résultats de l'expérience, dans la mesure où ils peuvent être confirmés, peuvent donc a posteriori jouer le rôle de repères objectifs, reproductibles ad libidum, qui sont d'une valeur considérable autant pour la formation de nouvelles générations de musicologues ou d'étudiants en musique, que pour la communication entre spécialistes ou encore entre spécialistes et non-spécialistes de la musique électroacoustique.

1.1.2 Identification de schémas récurrents

L'objectif de l'expérience est de parvenir à identifier quels algorithmes, quels paramètres et quelles plages de paramètres vont produire avec quelle certitude un signal audionumérique, puis électrique, puis un phénomène acoustique et enfin un phénomène de la perception qui sera reconnu comme membre de la catégorie étudiée.

La structure fonctionnelle de FLSC permet justement une comparaison entre ces données informatiques, puisque l'espace des fonctions employées définit en soi une classification hiérarchique d'objets de synthèse regroupés en familles dont les algorithmes peuvent être communs sur divers niveaux.

1.2 Champ d'investigation

Bien que, potentiellement, il soit possible par cette méthode d'étudier un critère arbitraire dans n'importe quelle théorie fondée sur un découpage catégoriel des phénomènes sonores, nous avons dû ici, par faute de temps, nous contenter d'analyser un seul critère de la théorie Santosienne.

Nous décomposons donc les différentes étapes qui mènent de la généralité de cette théorie d'inspiration Schaefferienne jusqu'à la situation particulière qui nous occupe.

1.2.1 Critère de matière

Pierre Schaeffer, comme de nombreux autres auteurs (en premier lieu Aristote), distingue notamment les critères de *forme* des critères de *matière*, ce qui revient comme nous l'avons décrit ci-dessus à différencier le contenu spectral de son évolution au cours du temps. Il existe également dans la théorie Schaefferienne des critères d'*entretien*, qui sont situés dans un entre-deux où l'échelle temporelle est à la limite du discernable, mais il est possible d'ignorer cette troisième catégorie si l'on admet qu'il s'agit d'une question de nuance.

Les critères de formes ayant été largement étudiés par ailleurs (et pour commencer dans l'analyse musicale classique qui en fait son principal objet), nous avons choisi de nous concentrer sur la matière, c'est à dire l'élément proprement sonore et concret qui fait la spécificité de la musicologie électroacoustique.

1.2.2 Sons de masse tonique

La théorie Santosienne distingue un grand nombre de catégories de *masse*, c'est à dire de sons dont le matériau sonore lui-même présente une caractéristique typologique singulière. Aux catégories de base de la théorie Schaefferienne que sont les masses toniques ou complexes, Jean-Louis Di Santo ajoute une décomposition supplémentaire des masses complexes en masses inharmoniques ou bruitées. Ces deux théories se poursuivent par l'analyse des masses *hybrides* (présentant des caractéristiques appartenant à plusieurs catégories distinctes), ainsi que des *groupes* (qui sont des sons composites qui constituent néanmoins une sorte d'unité).

Bien que nous puissions à l'heure actuelle émettre des hypothèses sur ce qui caractérise les sons bruités, ainsi que les sons toniques-bruités, et également certains types de sons inharmoniques, notre étude se résume à un critère caractéristique des sons toniques, puisque dans les deux théories il s'agit là de la catégorie la plus simple. Encore une fois, nous regrettons profondément le manque de temps qui nous a empêché de poursuivre plus avant ces études.

1.2.3 Critère riche-clair-voilé

En nous ramenant de la sorte à une seule catégorie de masse, dans le seul critère de matière, notre champ d'investigation se limite au critère de *timbre harmonique*, que nous avons identifié ci-dessus (cf. 1.5 page 59) à la notion d'amplitude relative des différentes composantes spectrales.

L'ensemble des fonctions permettant de moduler l'enveloppe spectrale d'un objet audionumérique étant potentiellement infini, avec des résultats très variés, il a encore une fois fallu restreindre plus avant les possibilités envisagées.

Selon Jean-Louis Di Santo, le critère le plus important est la distinction tout d'abord entre les sons *riches* et les sons *pauvres*, et à l'intérieur de cette dernière catégorie, celle entre les sons *clairs* et les sons *voilés*. Nous avons au passage envisagé d'autres distinctions entre des sons *chauds* ou *froids*, *brillants* ou *mats*, et certains *groupes*, sans pour autant disposer du temps nécessaire pour arriver à des conclusions en bonne et due forme.

1.3 Méthodologie

Hormis ses résultats, la valeur principale de cette expérience réside dans la méthode d'expérimentation elle-même, qui a été élaborée empiriquement au cours du processus et qui offre la possibilité d'être transposée dans différents contextes analogues, et éventuellement généralisée sous la forme d'un protocole expérimental.

Une observation distanciée de l'observation elle-même nous a permis de formuler certains points donnant lieu à réflexion, qui éclairent quelque peu cette démarche de recherche par tâtonnements successifs qui, bien qu'elle puisse sembler à première vue mal formalisée ou organisée, trouve finalement ses fondements dans une attitude strictement rationnelle.

1.3.1 Situation de double aveugle

Tout d'abord, il faut bien comprendre que nous sommes dans un cas de figure qu'on peut qualifier de situation de "double aveugle", dans la mesure où le musicologue (ici Jean-Louis Di Santo, qui a accepté avec enthousiasme de se prêter à une telle expérience) n'a pas de connaissances suffisantes pour bien comprendre les impératifs sous-jacents à la synthèse sonore vue sous l'angle des mathématiques, et où de façon converse nos connaissances et notre expérience en termes d'analyse musicale sont également largement insuffisants pour bien saisir les enjeux et les méthodes qui sous-tendent l'art difficile de l'analyse musicale.

Il s'agit donc d'un dialogue entre spécialistes de domaines très différents, réunis par la coïncidence de la musique par ordinateur, qui doit au travers d'un échange trouver un terrain commun sur lequel il est possible de construire une réalité commune : une mise en relation de ce qui pour l'un est une réalité audible, et pour l'autre une réalité numérique.

Il va sans dire que dans une telle entreprise, les échanges directs et une confiance sans faille sont de mise, ainsi qu'une grande capacité à remettre en question les divers a priori qui proviennent de notre domaine de compétence propre. Sur ce plan, et bien que cela ne soit pas le sujet de la présente publication, nous devons saluer l'ouverture d'esprit avec laquelle nos interlocuteurs ont pu accueillir des points de vue dont l'apparence était parfois contradictoire.

1.3.2 Recherche d'un consensus

Nous pouvons considérer comme un résultat – sous réserve de validation par un ensemble de personnes plus large – une notion, formulée de façon bilatérale, chacun suivant sa modalité propre, qui semble de façon stable dans le temps trouver un sens identique dans la perception de chacun.

En l'occurence, il s'agit pour nous de découvrir un algorithme, ainsi que la plage de paramètres associée, qui sera pour nous bien définie d'un point de vue informatique et qui correspondra systématiquement et de façon prévisible au résultat audible attendu par notre interlcuteur – et ceci *sans* qu'il soit nécessaire que notre interlocuteur comprenne dans les détails en quoi consiste cet algorithme ou ses paramètre, ni non plus que nous ayons besoin de comprendre comment il parvient à entendre ce qu'il entend, ou à comprendre ce qu'il comprend.

1.3.3 Boucle synthèse-écoute

Dans ce but, nous avons mis en place, de façon informelle, un protocole qui pourrait être décrit de la façon suivante :

- à l'issue d'une concertation, nous sommes en mesure de proposer un ensemble d'objets, produits par synthèse sonore, qui nous paraissent relever effectivement de la catégorie ciblée, sans pour autant être en mesure d'avoir la moindre certitude quant à la qualification de la sensation sonore devant y être associée
- ces objets peuvent ensuite être écoutés et analysés par notre partenaire, sans qu'il ait connaissance des moyens par lesquels ils ont été produits, et celui-ci peut donc nous indiquer comment les situer dans la théorie musicale, ce qui entraîne a minima qu'il peut confirmer, ou le plus souvent infirmer, qu'il s'agit bien d'une partie de l'ensemble recherché

Ce processus peut alors, au travers des observations que notre échange parvient à dégager, être itéré de nombreuses fois dans l'espoir d'aboutir au consensus recherché sur la correspondance entre l'algorithme audionumérique et le phénomène sonore associé au terme que nous cherchons à caractériser.

1.3.4 Analyse d'exemples

Au cours de notre expérience, cette démarche a également bénéficié d'être parfois inversée, notre partenaire proposant alors des exemples d'objets audionumériques associés à un phénomène sonore donné, que nous avons pu traiter par les méthodes de l'analyse du signal afin de découvrir en quoi consitait leur singularité.

C'est par cette méthode, et avec les apports du mémoire d'Anne-Emmanuelle Abrassart sur l'analyse spectrale des sons de flûte à bec [1], que nous sommes parvenus à produire des exemples de sons clairs, ce qui jusque là échappait à notre méthode expérimentale, malgré tous nos efforts pour extraire les critères de synthèse pertinents.

1.4 Résultats

Le résultat principal de cette étude est l'identification en termes de synthèse sonore de la distinction riche-clair-voilé formulée dans la théorie Santosienne ; c'est à dire un moyen de produire avec une certaine fiabilité des objets audionumériques produisant un percept associé à l'une de ces trois catégories, démontrant du même coup leur exclusion mutuelle.

1.4.1 Critère ternaire

Tout d'abord, nous avons découvert que ce critère est bel et bien un critère ternaire : les sons sont soit riches soit pauvres, et seulement lorsqu'ils sont pauvres, ils peuvent alors être clairs ou voilés.

La richesse d'un son tient essentiellement à la force relative dans sa composition spectrale des composantes de rang élevé (les partiels d'indice supérieur), ce qui associe à cette notion celle de *brillance* qui se définit d'une façon analogue.

Cet effet peut être modulé à loisir au moyen de la fonction de *dégressivité* décrite en 3.2.2 page 71. Une forte dégressivité produira plus facilement des sons pauvres.

Notre expérience tend à indiquer qu'il est absolument impossible qu'un son soit à la fois riche et clair, pour des raisons que nous exposerons dans ce qui suit.

1.4.2 Différence clair-voilé

La différence entre les sons clairs et voilés semble tenir essentiellement à l'impression plus ou moins forte produite par les battements associés à l'interaction entre les premiers partiels, lorsque ceux-ci sont fortement majoritaires (et donc que nous ne sommes pas dans le cas d'un son riche).

La fonction de *dominance*, décrite en 3.2.2 page 71, permet de fixer indépendamment le gain relatif affecté aux premiers partiels, ainsi que le nombre de partiels affectés ; il est possible de produire par cette méthode des sons clairs ou voilés, sachant que les sons les plus clairs se rapprochent finalement d'une simple sinusoïde.

L'effet est beaucoup plus net lorsqu'il est associé à une forte dégressivité ; la combinaison de la dominance et de la dégressivité constitue un critère déterminant dans la distinction riche-pauvre.

1.4.3 Caractérisation mathématique

L'expression mathématique de la composition spectrale des sons de synthèse associés est la suivante, pour un objet à N composantes de fréquence fondamentale f :

$$rcv(a,n,s) : t \mapsto \sum_{i=1}^{N} \quad sin(i \cdot f \cdot t) \cdot \tfrac{1}{i^s} \cdot 2^{a \cdot (n+1-i)|_0^1}$$

Dans ce cas, $s \in [1, +\infty[$ est l'indice de dégressivité, $a \in [0, +\infty[$ est le gain de dominance, et $n \in [0, N]$ est le nombre de partiels dominants. A noter qu'il est possible d'effectuer une transition continue entre les sons riches, clairs et voilés en exprimant ces paramètres comme des fonctions temporelles, ce qui devrait produire un son variant entre différentes catégories de timbre harmonique.

Sur la figure V.1, on peut voir les trois espaces de paramètres :

- les sons riches sont en rouge
- les sons clairs sont en vert
- les sons voilés sont en jaune

Les sons clairs n'existent que lorsque $n = 1$ et que a et s sont suffisamment élevés. Les sons riches sont ceux pour lesquels $a = 0$, ou alors a, n et s sont suffisamment faibles. Les autres sons sont voilés (lorsque $n > 1$ et a et s sont suffisamment importants).

L'étendue exacte des zones n'est pas déterminée précisément, et les mesures de la figure ne sont données qu'à titre indicatif. Il paraît difficilement envisageable de définir ce critère de façon strictement quantitative, les cas intermédiaires étant le plus souvent des entre-deux.

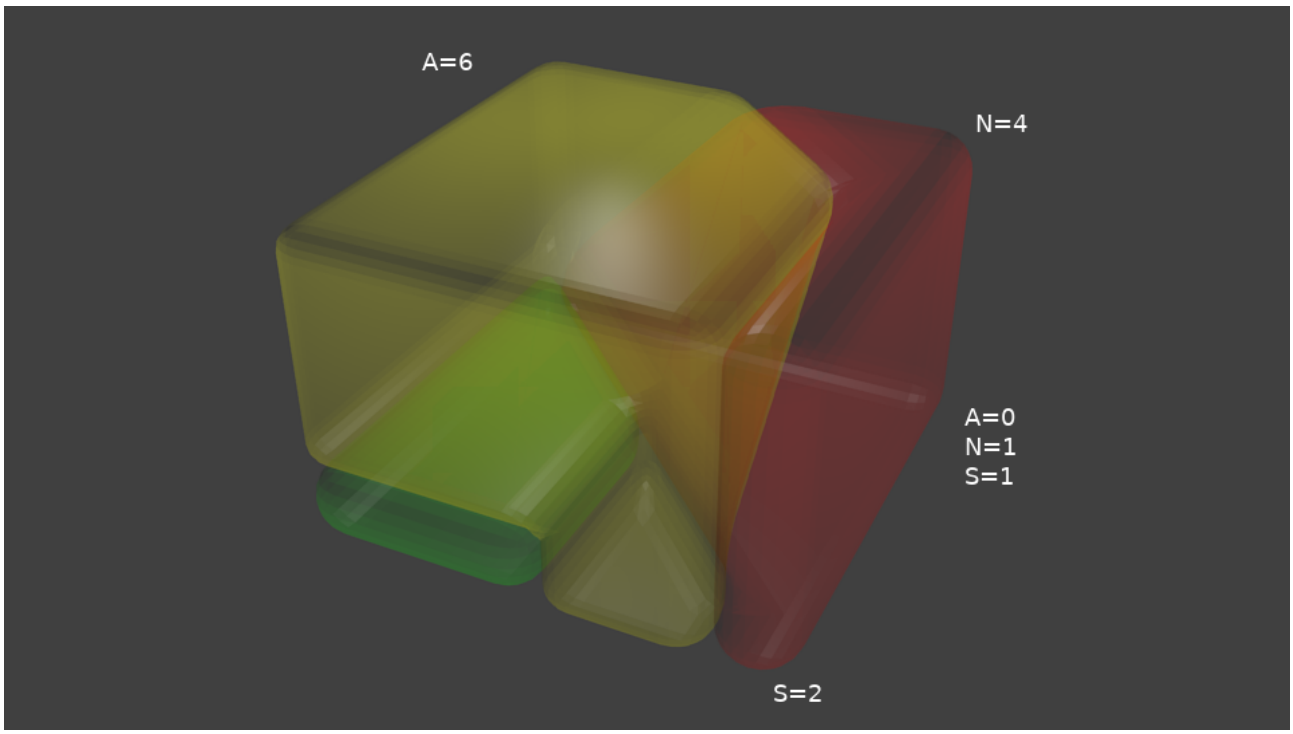

FIGURE V.1 – Espaces de paramètres des catégories riche-clair-voilé

1.5 Perspectives

Cette étude, bien que longue et laborieuse, débouche sur un résultat concret dont la validité est accréditée notamment par le fait qu'au cours de ce processus, il a été possible de vérifier la cohérence du modèle et de la perception par des essais sur des enregistrements sans indication de contenu et rangés dans un ordre aléatoire.

Nous pouvons donc envisager d'évaluer le caractère général de ce résultat, par des expériences sur un public plus large ; et dans le même temps, de poursuivre dans la recherche sur d'autres critères ou sur d'autres théories, en généralisant ce mode de fonctionnement.

1.5.1 Validation psychoacoustique

Le fait qu'un individu donné soit effectivement capable *en aveugle* (c'est à dire en raisonnant uniquement sur ses perceptions auditives) de retrouver la correpondance prédite entre les sons de synthèse et les catégories de la théorie démontre a minima que les manipulations numériques en question produisent des résultats discernables par l'audition, dont les schémas de variation se comportent de manière uniforme.

Nous savons par ailleurs que la culture des individus joue un rôle important dans leur appréciation des phénomènes, et particulièrement lorsqu'il s'agit de musique. La question qui se pose, donc, est de découvrir dans quelle mesure d'autres personnes, dont l'expérience dans le domaine de la musique peut varier, pourront reproduire des résultats analogues.

Sur ce point nous avons l'alternative, soit de proposer une tâche non dirigée afin d'observer l'ensemble des distinctions catégorielles émergeant d'une masse d'individus indéterminés, soit au contraire d'évaluer par une tâche dirigée la capacité de ces mêmes individus à apprendre à reconnaître les critères perceptifs que nous avons modélisés.

Ces deux approches produiraient des résultats de nature différente, et effectuer les deux en parallèle exigerait très certainement que les études soient menées sur des groupes distincts.

1.5.2 Autres critères

Au cours de cette recherche empirique, nous avons de façon annexe produit un certain nombre d'autres manipulations numériques dont les effets étaient potentiellement intéressants, bien que n'étant finalement pas l'objet de notre enquête.

Nous pouvons donc proposer d'autres expériences analogues, sur des notions de timbre ou sur d'autres types de masse, par exemple celles des bruits ou des sons inharmoniques.

Bien évidemment, ceci est essentiellement une question de temps et de moyens humains, car comme nous avons pu le constater ce processus exige un investissement considérable de la part des participants.

1.5.3 Généralisation de la méthode

Par ailleurs, la méthode de travail que nous avons construite, ainsi que les observations que nous avons pu faire dans ce temps, pourraient donner lieu à une analyse plus approfondie dans le but de formuler une méthodologie plus générale permettant de faire face à ce type de situation.

Outre la recherche musicale, il est probable que ce type de problème existe dans de nombreux autres domaines, lorsqu'il s'agit d'établir la relation entre un modèle informatique et l'expérience humaine. Il s'agit en quelque sorte d'une méthode de transcription informatique des connaissances d'un expert, dans les cas où il est difficile d'envisager des méthodes d'apprentissage automatique.

2 Expérience SNDIFF

Le travail de recherche décrit ci-dessus (cf. 1 page 73) nous a conduit à une tentative de vérification formelle de la validité du critère mis en évidence, au travers d'une expérience destinée à un ensemble de sujets volontaires et portant sur la notion de différence qualitative entre des objets audionumériques choisis aléatoirement parmi un ensemble prédéfini, présentant les critères dont nous cherchions à vérifier un éventuel découpage catégoriel présent naturellement dans l'audition humaine. Cette expérience portant sur la notion de différence entre les sons, elle fût nommée en conséquence *SNDIFF* [19].

Notre collaboration avec Catherine Semal, psychoacousticienne, aura été d'une importance capitale dans la formulation d'un protocole expérimental adéquat, muni d'une méthodologie permettant un résultat réellement conclusif. Malheureusement pour nous, certains facteurs humains comme l'investissement important demandé au public ont eu pour conséquence l'échec de cette expérience dont le contenu avait pourtant été spécifié avec le plus grand soin.

2.1 Description de l'expérience

Tout d'abord, nous exposons ici la nature de cette expérience : l'objectif recherché, le principe de son fonctionnement, et les moyens dont nous avons pu disposer.

Malgré l'absence de résultat, la démarche scientifique que nous avons entreprise reste fondamentalement valide (exception faite des facteurs non-contrôlés qui sont toujours présents malgré toute précaution, ce qui est un phénomène généralisé en neuroscience). Il nous semble donc pertinent d'expliquer autant que possible nos choix protocolaires, qui peuvent néanmoins servir de base à de futures tentatives expérimentales.

2.1.1 Objectif

Le but de l'expérience est de vérifier la pertinence de la distinction riche/clair/voilé identifiée par Jean-Louis Di Santo, en évaluant la propension d'un public de sujets (environ 25) à effectuer une différenciation analogue, apparaissant de façon stochastique dans une tâche consistant à évaluer la disparité relative de sons présentant une variation sur les critères de synthèse que nous avons pu associer (ainsi que sur d'autres, de façon à pouvoir évaluer la primauté du critère sur d'autres critères de synthèse).

L'hypothèse initiale est que moyennant une carte des distances parmi un ensemble de sons, il nous est possible de déterminer quelles sont les parties de cet ensemble qui s'aggrègent autour de polarités définissant des sons types d'une catégorie associée.

2.1.2 Principe

La méthode choisie, qui est une méthode connue dans le champ de ce type d'expérimentation, consiste à effectuer de façon répétée la tâche élémentaire suivante : moyennant trois éléments de la base de sons, choisis aléatoirement, indiquer lequel des trois semble le plus différent des deux autres.

Ce procédé permet d'éviter l'écueil de devoir évaluer un "degré de différence" entre deux sons, ce qui peut constituer un problème insoluble dans la plupart des cas. De même si la question se résume à devoir déterminer si deux sons sont similaires ou différents, il est probable qu'elle soit le plus souvent sans réponse satisfaisante.

Bien évidemment, il n'est pas possible par ce procédé de déterminer si les trois sons proposés étaient globalement proches ou éloignés, mais la supposition est que ce facteur est éliminé en tant que bruitage décorrélé au cours de la résolution statistique des observations.

2.1.3 Moyens

Afin de mettre en place cette expérience, il nous a tout d'abord fallu déterminer un ensemble de sons aussi réduit que possible, tout en conservant une diversité suffisante. Notamment, étant conscients de la sensibilité du résultat à la hauteur absolue des sons employés (les critères spectraux étant potentiellement perçus de façon différente suivant la fréquence des composantes), nous avons dû maintenir une diversité des fréquences de base afin de minimiser ce phénomène. De même, deux autres critères de synthèse, la *présence cyclique* et la *présence logarithmique* (3.2.2, 3.2.2), dont nous avions pu consatater l'importance, ont été intégrés à cette diversité.

La production de ces sons au moyen de la bibliothèque FLSC employée avec Jean-Louis Di Santo n'a posé aucun problème particulier, étant donné que l'implémentation est munie d'un système permettant la génération d'un catalogue de sons par épuisement combinatoire des ensembles de paramètres.

Ensuite, nous avons pu proposer l'expérience par le biais d'un site Web minimal, programmé par nos soins (en PHP/HTML), qui permettait de recueillir les informations nécessaires (âge, sexe, pays d'origine, expérience préalable de la musique) et les réponses aux instances de la question proposée, ainsi que de donner les indications préalables nécessaires et de proposer aux personnes ayant mené l'expérience à son terme de s'inscrire à une liste de diffusion publiant les résultats.

L'annonce de l'existence de l'expérience a été effectuée par les réseaux de communication auquels nous avions accès ; il est possible que notre impact ait été quelque peu limité, mais dans le même temps il est évident qu'il existe à l'heure actuelle une telle quantité d'informations pertinentes qu'une initiative d'une importance aussi faible doit prendre en compte la faible disponibilité du public.

2.2 Déroulement et conséquences

Après deux mois de mise en ligne du site Web de l'expérience, le bilan était déjà extrêmement clair : environ 20 ou 25 personnes en tout avaient débuté l'expérience, mais seulement 8 l'avaient menée à son terme (en incluant trois d'entre nous qui étions à son origine).

Il nous est méthodologiquement impossible de prendre en compte des réponses partielles, étant donné que l'expérience de la tâche est un facteur déterminant dans l'efficacité des sujets. En conséquence, le nombre de réponses trop faible ne nous permet pas d'espérer obtenir une interprétation fiable.

Il semble a posteriori que bien que la tâche proposée ait été réduite à sa plus simple expression, elle reste trop longue et trop difficile pour un public dont la motivation et l'expérience est insuffisante. Nous pouvons néanmoins tenter de tirer les leçons de cet échec, et éventuellement adapter la méthode dans l'espoir d'obtenir un résultat moins fort mais pratiquable.

2.3 Perspectives résultantes

Nous examinons ici les différentes stratégies de récupération qui pourraient permettre d'obtenir un résultat sur le même thème, bien qu'il semble actuellement impossible de les mettre en œuvre pour des questions d'organisation du travail.

2.3.1 Possibilité de simplification

Nous pourrions envisager de simplifier la tâche proposée (200 essais sur une base de 2500 sons) pour la ramener à un nombre d'essais beaucoup plus réduit (environ 50), ce qui entraîne de limiter fortement le nombre de sons (environ 500).

Néanmoins, ceci introduit des biais méthodologiques, dont au moins deux semblent évidents :

- Limiter à ce point la diversité des sons suppose de supprimer tous les critères n'étant pas l'objet direct de l'expérience ; en conséquence, il serait impossible de mesurer l'importance du critère de synthèse par rapport à celle d'autres critères. Nous serions donc dans le cadre d'une expérience totalement dirigée qui ne permettrait pas de vérifier le caractère naturel de la distinction. De plus, en éliminant les différences de hauteur, il n'est pas acquis que le résultat soit indépendant de la hauteur des sons, ce qui est fortement questionnable.
- Sur une série d'essais aussi courte, la plupart des sujets n'auront pas le temps de construire leur propre expérience de la tâche requise, et il est donc probable que les réponses seront essentiellement aléatoires ; ceci peut nous amener à une observation négative (le phénomène espéré ne se produit pas) alors que des sujets ayant plus d'expérience auraient produit un résultat positif, leur pouvoir de différenciation s'étant accru au cours de l'expérience.

Il nous semble donc que de simplifier cette expérience, que nous avions déjà formulé de la façon la plus simple possible, ne donnerait aucun résultat satisfaisant.

2.3.2 Possibilité de renforcement des moyens

Une autre possibilité consiste à répéter l'expérience, mais cette fois en faisant appel à des sujets beaucoup plus expérimentés, et motivés par une rémunération. Dans ce cas il est possible d'obtenir des résultats solides, bien que le nombre de sujets disponibles étant réduit, cela entraîne le biais de fonder nos résultats expérimentaux sur une population qui représente d'autant moins le cas général.

Il est relativement clair dans notre cas que le coût économique engagé est démesuré eût égard à l'importance minime du résultat ; comme nous le mentionnons ci-dessus, cette étude est simplement une instance d'un grand nombre d'études qui pourraient être menées sur le même schéma, et il faudrait donc répéter ce processus de nombreuses fois avant d'obtenir des résultats dignes d'intérêt. Bien entendu dans ce cas, l'investissement humain nécessaire serait un facteur écrasant.

2.3.3 Reformulation du problème

Finalement, les seules solutions à notre portée semblent devoir respecter impérativement la contrainte de reposer sur une tâche beacoup plus simple, et accessible à un public très large avec un investissement individuel minime.

En l'occurence, il serait possible de demander au public de travailler sur un nombre très restreint de sons type, et de les séparer itérativement à chaque étape en deux parties distinctes, jusqu'à ce qu'il n'en reste qu'un seul par partie ; ceci permettrait au moins de comparer statistiquement la distance entre les sons types.

La contrepartie de cette restriction est que ce type d'expérience ne donnerait que des résultats approximatifs, dont la validité formelle serait quasi-nulle ; ils ne pourraient guère servir que d'indications générales permettant d'émettre de nouvelles hypothèses pouvant orienter la recherche musicale.

2.4 Ouverture

De notre point de vue, cet échec dans une expérience visant à combiner la méthodologie de la psychoacoustique avec des conceptions musicologiques semble indiquer qu'il serait nécessaire de mieux définir la méthodologie de la recherche musicale.

Comme Schaeffer avait pu l'observer [7], les questionnements d'ordre musical portent sur des phénomènes qui sont d'emblée complexes au regard de ceux qui sont suffisamment élémentaires pour être objets d'une étude sérieuse d'un point de vue psychoacoustique ; peut-être alors que la notion de validité formelle n'est pas une priorité d'un point de vue musicologique ?

Nous devons au passage constater que la culture des arts numériques évolue à l'heure actuelle à une allure telle qu'il est possible que les vérités d'aujourd'hui tombent en désuétude dans un avenir proche ; peut-être alors devrions nous simplement chercher des pistes, et non des conclusions ?

Quoiqu'il en soit, l'expérimentation sur les critères de la perception nous permet de proposer de nouvelles techniques permettant la synthèse de sons inouïs. Nous espérons ainsi donner de nouveaux moyens à la création artistique ; il est possible que l'interprétation scientifique de ces tentatives n'arrive qu'a posteriori d'une longue expérience culturelle.

3 Sonify

La tentative de démarche de science participative intitulée *Sonify* – jeu en ligne de création et de classification d'objets de synthèse sonore – se situe dans le contexte de l'échec manifeste de la stratégie expérimentale classique sur l'identification des critères musicologiques dans leur relation aux critères de synthèse audionumérique.

En l'occurence, le constat réside dans le fait que :

- le nombre d'hypothèses à évaluer est beaucoup trop grand ; pour être impartiaux, il nous faudrait évaluer chaque critère de chaque théorie musicologique, or chaque théorie comporte un certain nombre de critères et il existe autant de théories que de musicologues ; nous sommes donc devant une complexité écrasante
- le travail nécessaire pour évaluer ne serait-ce qu'une seule de ces hypothèses est déjà au delà de nos moyens, étant donné que l'engagement nécessaire de la part du public semble totalement inatteignable, dans la mesure où les priorités de la recherche en général sont à réserver à des sujets bien plus critiques

Ceci nous conduit à suggérer une reformulation radicale de la méthodologie de la recherche musicale, et à tenter de proposer des pistes pour de nouvelles approches inspirées par celle-ci qui, nous l'espérons, permettront d'effectuer des progrès là où la méthode classique ne peut qu'échouer.

3.1 Remise en question du paradigme

D'un point de vue scientifique, il se pose la différenciation suivante : celle de la rationalisation *a posteriori* opposée à une rationalisation *en temps réel*, c'est à dire au fur et à mesure de l'apparition des phénomènes étudiés.

Alors, on doit séparer la recherche d'une loi statique, dérivant de la nature, de la notion de loi dynamique, qui serait une conséquence de la culture, et donc une vérité contextuelle et non pas une vérité relative à l'humanité dans son ensemble, au travers des époques et des cultures.

Etant donné l'importance de la culture dans l'identification des sons, en particulier lorsqu'il s'agit de musique (donc de son à finalité purement esthétisante), il semble primordial de mettre en place une méthodologie qui puisse faire la différence entre le local et le global.

En effet, on ne saurait traiter des phénomènes culturels en évolution continue par la méthodologie des sciences expérimentales : le résultat de telles expérimentations ne pourrait en aucun cas être pris pour un grand déterminisme de la nature humaine, mais bien au contraire comme un indicateur d'une vérité "ici et maintenant".

Ceci déplace la question scientifique, depuis le rapport d'une connaissance accumulée sur la complexité inhérente du problème, vers le rapport d'une capacité d'apprentissage à l'instant donné sur la capacité de nouvelles créations à ce même instant.

Dans la mesure où *a priori* tout individu est créateur culturel, ce problème ne peut être traité que par le développement d'une autonomie des individus dans l'analyse simultanée de leurs propres productions, ainsi que de celles qui les environnent.

Ce positionnement nous conduit à proposer à un public aussi large que possible un outil de création et de classification des objets audionumériques, en jouant autant que possible sur l'aspect ludique et libérateur d'une telle pratique.

3.2 Présentation du projet

L'analyse du besoin nous conduit à décomposer le projet suivant deux axes complémentaires : les objectifs poursuivis et les moyens mis en œuvre. Puisqu'il s'agit ici d'une tentative innovante, les objectifs doivent apparaître le plus clairement possible tout en minimisant autant que possible les moyens mis en œuvre, dans l'espoir que la démarche puisse par la suite inspirer d'autres initiatives allant dans le même sens.

3.2.1 Objectifs

Le but de la démarche se décline suivant deux directions converses :

- d'une part il s'agit de mettre à la disposition du public un outil permettant la création sonore, afin de démystifier cette activité et ouvrir la voie à une compréhension enactive de la nature des sons artificiels
- d'autre part il est nécessaire, par une démarche collective, de recueillir toutes les informations nécessaires à la situation, les uns par rapport aux autres, des artefacts audionumériques produits au cours de cette activité

En effet l'expérience produite par ce mouvement reste sans valeur scientifique si il n'est pas possible de rapporter les productions des membres les unes aux autres de façon à pouvoir établir les invariants qui rangeront celles-ci dans des catégories pouvant prétendre à l'universalité.

Au cours de ce processus, la relation entre *faire* et *entendre ce que l'on fait*, chère à Pierre Schaeffer, semble préservée dans la mesure où l'activité des membres n'a de valeur que dans la mesure où elle peut être située par les autres membres, au sein de l'univers sonore produit.

3.2.2 Moyens

La solution matérielle la plus adaptée nous a semblé résider dans le couplage d'un moteur de synthèse sonore, créé par nos soins suivant notre expérience (et dont le code source est donné en B page 107), avec une plateforme Web permettant l'accès à un public aussi large que possible. Cette dernière fut réalisée entre avril et septembre 2018 par Axel Camus, stagiaire de M2 "Image & Son" de l'Université de Bordeaux [3].

La plateforme prend l'aspect d'un jeu, dans lequel nous proposons aux joueurs de créer des sons, suivant les paramètres du synthétiseur que nous avons produit, mais sans toutefois leur demander de comprendre les tenants et aboutissants de ce logiciel. Au contraire, il s'agit pour eux d'expérimenter *en aveugle* et *à l'oreille* afin d'obtenir les résultats qu'ils ou elles souhaitent obtenir.

Afin d'éviter une explosion du nombre de sons proposés, la capacité des joueurs à produire de nouveaux sons au cours du temps est conditionnée par la façon dont ils ou elles classifient les sons de la base partagée : un

système de suggestion fondé sur les choix des joueurs sert également à les rétribuer (en termes de capacité de production), lorsque leurs choix semblent constituer une indication valide pour les choix des autres. Ce procédé semble promouvoir une méritocratie de la pertinence des choix de classification, et constitue l'enjeu de l'activité.

3.2.3 Interêt du public

La base de sons produits est placée sous une licence libre, qui reconnaît la paternité de l'œuvre et autorise l'usage non commercial et la réappropriation, à condition de propager cette même licence. Les usagers peuvent sans autre contrainte télécharger les sons produits par eux-mêmes ou par d'autres, lorsqu'ils parviennent à les localiser dans la base de sons ; cette tâche est facilitée par des efforts de classification pertinents.

Ainsi, le jeu se destine éventuellement à des artistes en recherche de sons pouvant alimenter leur travail, mais également à des particuliers ayant un besoin en termes de design sonore (notamment, il est probable que les sons produits font bon usage en tant que sonneries de téléphone).

Ce procédé de recherche collective de sons pertinents, alimenté par le système d'indexation dont le fonctionnement est également collectif, est présumé susceptible de produire un engagement suffisant de la part des membres potentiels pour que la démarche puisse prendre de l'ampleur. Comme toujours dans ce type de démarche, la stratégie de communication mise en place sera déterminante dans le démarrage du projet.

3.2.4 Résultats espérés

Ce projet permet d'escompter divers résultats, qui présentent chacun un intérêt indépendant du point de vue de la communauté artistico-scientifique :

- Premièrement, il produit *per se* une base de sons de synthèse, et les informations de classification afférentes, qui sont placées sous une licence libre et partageable et qui pourront donc être réutilisées à loisir par des professionnels du son, ainsi que par des particuliers ; il est donc possible de voir l'émergence de nouvelles pratiques de création sonore, au travers du partage de fragments d'enregistrements audionumériques. Les créations étant stockées sous la forme de code source et non nécessairement d'enregistrements, la capacité du serveur est virtuellement illimitée.
- La classification produite, associée aux paramètres de génération des sons, ouvre la possibilité à des tentatives de *fouille de données* qui pourraient révéler les invariants déterminant l'appartenance d'un son de synthèse à une catégorie humainement perceptible, suivant ses paramètres. A supposer qu'il soit possible de distinguer les biais de la représentation choisie des invariants de l'audition humaine, il serait donc possible de corriger le modèle afin de le faire correspondre au plus près aux *paramètres perceptifs*, c'est à dire aux variables observables dans le signal sonore qui qualifient le son d'un façon universelle, dans la perception humaine.
- Le simple fait de proposer en parallèle la tâche de création et la tâche d'analyse, à un public aussi large que possible, constitue une transmission des savoirs-faire élémentaires de la composition électroacoustique, jusque là mis en œuvre par une petite minorité, vers un ensemble de personnes beaucoup plus large ; le bénéfice culturel de cette démystification pourrait avoir un impact considérable sur l'évolution de la musique par ordinateur, à supposer que la démarche puisse être reproduite et améliorée. La démocratisation de la synthèse sonore, ainsi, ne serait qu'une question de mise à disposition d'outils favorisant l'intuition et l'écoute par rapport à la compétence analytique.

4 Conclusion

Cette série d'expériences, bien que méthodologiquement fondée, n'a pas à l'heure actuelle donné de résultats conséquents ; la question des moyens humains disponibles semble *a posteriori* prédominante, étant donnée la masse de travail devant être accomplie.

Néanmoins, la confrontation avec la réalité sociale nous a conduit à évoluer dans notre approche du problème, et à tenter de donner des pistes en direction d'une recherche musicale qui reposerait sur un travail collectif, dans lequel chacun trouverait son intérêt.

Il nous semble au travers de cette démarche proposer une approche innovante, du type de la science participative, dont la nature ouvrirait le champ à une connaissance moins centralisée, plus contextuelle, et beaucoup plus vivace de l'art difficile de la compréhension du sonore.

Bien entendu, ce projet ne reste qu'une ébauche des nouvelles possibilités qui nous sont offertes, mais nous pensons qu'il est possible par ce biais d'effectuer le dépassement de la limitation évoquée ci-dessus d'une méthodologie strictement expérimentale, et visant des résultats statiques. Au contraire, il s'agit ici d'un processus purement dynamique qui met à la portée du public, et valorise, le *travail* scientifique au-delà des *résultats* scientifiques, notre croyance étant que la science est un processus, et non un produit.

Chapitre VI

Bilan & perspectives

Pour synthétiser le travail effectué, celui-ci se compose suivant trois axes complémentaires :

- Une recherche fondamentale qui définit la synthèse sonore de partitions virtuelles comme un langage formel, complet, minimal et pratiquable (II et III).
- Une application pratique qui développe un système de synthèse additive extensible et fondé sur les critères perceptifs de la théorie Schaefferienne (IV).
- Une démarche expérimentale qui propose des pistes pour l'identification, la validation, et la transmission des associations entre critères de synthèse et critères perceptifs que nous pouvons formuler au moyen de ces outils théoriques et pratiques (V).

Il s'agit donc d'une démarche qui tente de mettre en synergie ces trois choses fondamentales à la musique numérique que sont la science, la technologie et la culture.

Chacune de ces directions peut se poursuivre au travers des nombreuses portes ouvertes par cette tentative initiale ; nous en présenterons un certain nombre en section 2.

Mais pour commencer, nous allons en section 1 dresser l'inventaire des concepts, outils, méthodes et expériences qui constituent l'ossature de ce travail.

1 Structure logique

Nous présentons ici une liste logiquement structurée des différents éléments qui constituent le système, avec les commentaires qui décrivent leur nature. Elle donne une vue synthétique du travail effectué et de son organisation.

1. **Théorie des partitions virtuelles sonores** : une modélisation complète des partitions virtuelles incluant les critères sonores, assortie d'une implémentation.
   (a) **Outils mathématiques** : les outils mathématiques employés.
      i. **Temps** : une représentation du temps, des bases de temps et des séquences de dates.
      ii. **Signaux / Fonctions temporelles** : une représentation des fonctions qui dépendent du temps, c'est à dire les signaux audibles et les signaux de contrôle.
   (b) **Temporalité souple** : un modèle du temps permettant la spécification abstraite et la distorsion de structures qui en dépendent.
      i. **Valeurs de temps déformable** : un modèle de date permettant la distorsion par la composition de fonctions numériques.
      ii. **Structures de temps abstrait** : un modèle permettant d'exprimer des intervalles et des séquences d'intervalles de temps déformable.
      iii. **Objets temporels** : un modèle permettant d'exprimer un objet audionumérique en faisant abstraction du temps.
      iv. **Réalisation des abstractions temporelles** : un système permettant de ramener les abstractions temporelles à un ensemble de dates concrètes et réalisables.
   (c) **Opérations temporelles** : un ensemble minimal d'opérations offrant une expressivité totale des distorsions possibles sur les abstractions temporelles.
      i. **Positionnement temporel** : les opérations permettant de positionner les valeurs temporelles, de façon absolue ou relative.

ii. **Composition d'objets** : les opérations permettant d'effectuer des assemblages d'objets temporels.

iii. **Cohérence temporelle** : les règles permettant de vérifier le maintien de la cohérence au travers des opérations temporelles.

(d) **Primitives de synthèse** : un ensemble minimal d'objets élémentaires nécessaires à la construction de structures audionumériques de complexité arbitraire.

i. **Générateurs spectraux** : les éléments permettant la génération d'un signal audible, condition nécessaire et suffisante à l'existence d'un objet audionumérique.

ii. **Contrôles temporels** : les éléments nécessaires au paramétrage des objets audionumériques, c'est à dire à l'expression de leur morphologie.

(e) **Langage formel** : les formes particulières nécessaires et suffisantes à l'opération d'un dialecte fonctionnel d'expressivité complète, et ainsi à la formulation de structures audionumériques arbitraires.

i. **Fonctions** : le mécanisme permettant la substitution de variables dans une sous-expression, qui est fondamentale au λ-calcul.

ii. **Conditionnelles** : le mécanisme permettant d'obtenir une variance dans le graphe d'exécution d'un programme.

iii. **Récursivité** : le mécanisme permettant d'exprimer des fonctions récursives, qui avec les conditionnelles permet de porter le λ-calcul au niveau d'un langage totalement expressif.

(f) **Implémentation** : la réalisation concrète du système formel défini, qui constitue une preuve de sa cohérence, et permet l'expérimentation avec ses éléments.

2. **Système de synthèse Schaefferienne** : un système complet pour la synthèse sonore selon des critères perceptifs, qui en conjonction avec le formalisme et son implémentation, permet de produire un outil de création sonore d'une expressivité considérable.

(a) **Décomposition de l'objet** : une analyse de la théorie Schaefferienne permettant d'exprimer les objets audionumériques comme structures composées de différents éléments complémentaires.

i. **Forme** : les éléments qui permettent de définir l'objet dans ses variations temporelles humainement discernables.

A. **Temporalité** : une représentation de la disposition des objets au cours du temps, qui est indispensable à la notion de forme.

B. **Paramètres** : une représentation de l'évolution temporelle des valeurs qui définissent les objets, qui est l'autre élément qui définit la forme.

ii. **Matière** : les éléments qui permettent de définir le matériau sonore constitutif de l'objet, dans sa nature spectrale et donc homogène du point de vue humain.

A. **Masse** : la disposition des composantes dans l'espace spectral.

B. **Timbre** : les relations d'intensité entre les différentes composantes.

(b) **Forme générale** : la façon dont les différentes composantes d'un objet peuvent être assemblées afin de le synthétiser, ce qui donne lieu à un algorithme général permettant, à partir des fonctions associées aux composantes choisies, de synthétiser n'importe quel objet formulé d'une façon compatible avec le système.

(c) **Typologie** : un catalogue extensible contenant les définitions des fonctions algébriques permettant de modéliser l'ensemble des comportements dont nous suggérons qu'ils pourraient être pertinents, dans le cadre d'un tel système de synthèse, ce qui permet de produire une grande diversité d'objets définis par des catégories numériques et potentiellement perceptives.

3. **Expérimentation musico-psycho-sociale** : un premier parcours expérimental visant à l'origine à identifier des hypothèses concernant les critères perceptifs, puis à évaluer leur pertinence d'une façon plus générale. Au cours de ce processus, de nombreux questionnements ont conduit à une réflexion critique sur la nature même de la recherche musicale, et sur les protocoles méthodologiquement adéquats dans ce domaine.

(a) **Critère Clair/Voilé/Riche** : un dialogue long et réactif avec une personne experte en musicologie, dans une démarche de recherche visant à identifier et modéliser les critères de la théorie Schaefferienne. Au cours de ce processus s'est posée la question de la singularité individuelle des percepts musicaux.

(b) **SNDIFF** : une expérience selon les méthodes de la psychoacoustique, sur un panel de sujets volontaires, ayant pour but de vérifier la validité des résultats précédents sur un certain nombre d'individus. Il est apparu un questionnement sur l'impact de la culture et de l'apprentissage dans la reconnaissance de critères perceptifs, et donc sur la possibilité de distinguer le naturel du culturel.

(c) **Sonify** : une ouverture vers la science participative, destinée à un large public, qui tente de construire de façon communautaire une vérité contextuelle de la classification perceptive des objets audionumériques. L'interrogation associée est la possibilité de l'émergence d'une culture massive de la synthèse sonore, par le biais d'une transmission horizontale des savoir-faire.

2 Développements possibles

Le projet dans ses différents aspects (théorique, applicatif, médiatique) ouvre un grand nombre de possibilités, en tant que système minimal ouvert au développement. Nous proposons ici, suivant ses axes, les perspectives de continuation que nous pouvons d'ores et déjà envisager, sachant qu'il existe probablement beaucoup de pistes qui nous ont échappé et nous apparaîtront peut-être plus tard.

2.1 Théorie, langage et implémentation

Les avancées possibles au niveau du langage lui-même peuvent se décomposer suivant qu'elles concernent le modèle en propre, ou bien son implémentation. Le modèle fournit un point de départ intéressant qui pourrait être étendu à d'autres domaines connexes. L'implémentation, quant à elle, est encore expérimentale et pourrait bénéficier d'un certain nombre d'améliorations qui en réduiraient les limites.

2.1.1 Modélisation

A l'heure actuelle, le modèle ne traite que de partitions statiques, il ne permet qu'un rendu monocanal, et bien entendu il ne produit que des phénomènes audionumériques. Néanmoins ces limitations peuvent être dépassées, étant donné que sa base, la logique temporelle, peut être étendue ou transposée de plusieurs façons.

Temporalités avancées :
En l'état, le modèle temporel permet les distorsions (et donc une notion de *temps souple*) sur des structures temporelles abstraites, mais il ne s'applique que hors temps réel, sans entrées utilisateur, sans réactivité et de façon purement déterministe. Il est pourtant envisageable d'étendre la notion de temporalité de façon à adresser ces potentiels, dans l'ordre suivant :

1. Tout d'abord, le système peut être adapté à l'exécution *en temps réel*, de sorte que le programme soit évalué de façon progressive au fil de son exécution. Ceci permettrait d'écrire des partitions potentiellement infinies, c'est à dire des processus musicaux non-terminaux. Pour cela, une première approche consiste à modifier l'opérateur `delay` pour y ajouter la notion d'*excécution différée*, c'est à dire que lorsqu'il est rencontré, l'exécution est suspendue jusqu'à ce que le retard demandé soit écoulé [1]. Alors il est possible d'écrire des algorithmes non-terminaux, à condition que chaque branche atteigne régulièrement un point d'arrêt.
2. Il est relativement facile d'ajouter des primitives `input` permettant des entrées de la part d'autres processus parallèles, qu'il s'agisse de flux audio ou de contrôle ; celles-ci pourraient prendre la place d'un module dans le graphe de traitement. Dans ce cas les objets spécifiés deviendraient interactifs, agissant comme des filtres ou des instruments virtuels.
3. Il est envisageable, dans la mesure où le système d'évaluation est temporalisé, de mettre des actions en attente non pas d'un temps écoulé, mais d'un évènement extérieur agissant comme un déclencheur. Un opérateur `trigger`, analogue de `delay`, pourrait jouer ce rôle en prenant en paramètre non pas une durée mais une adresse sur laquelle écouter. Dans ce cas le système de temps souple deviendrait de plus réactif, non seulement au niveau des objets mais au niveau de l'exécution elle même. De cette façon, le processus pourrait être par exemple un synthétiseur lié à un clavier maître.
4. Finalement, la dernière étape consiste à rendre l'exécution elle-même *non déterministe*, c'est à dire que des conditionnelles dans le graphe de calcul pourraient dépendre de valeurs passées en entrée par l'utilisateur. Il peut y avoir plusieurs façons de le mettre en œuvre, mais la plus simple serait de proposer un opérateur `latch` permettant de capturer une valeur de signal (probablement plutôt un contrôle) et de renvoyer le nombre correspondant. Les autres éléments de temporisation et de déclenchement permettraient alors de spécifier à quel moment la valeur doit être capturée, et ainsi de construire des structures temporelles non déterministes complexes avec des conditionnelles, des boucles et des fonctions récursives.

1. En réalité la question est légèrement plus complexe, étant donné que l'exécution n'est pas instantanée ; il faut également anticiper le temps d'éxécution nécessaire. Par exemple, l'étape suivante pourrait être précalculée, jusqu'à un nouveau point d'arrêt.

Spatialisation :

Une autre lacune profonde du modèle est qu'il ne traite à l'heure actuelle que de signaux monocanal, la sortie audio étant elle-même monophonique, implicitement et de façon immuable. Néanmoins ceci n'est pas nécessairement difficile à modifier ; il s'agirait essentiellement de proposer un ensemble minimal de primitives pour la répartition et le routage des signaux multicanal, et d'ajouter dans la notion de variable de liaison (les *bus*, qui sont en réalité le type représentant la valeur instantanée des objets audionumériques) l'information du nombre de canaux. Ensuite la sortie du programme tout entier pourrait s'adapter au nombre de canaux présents au retour de l'expression globale.

Modalité visuelle :

Un des défis les plus intéressants, finalement, serait de mettre la logique temporelle et la notion de paramètre en œuvre pour manipuler des spécifications autres qu'audionumériques ; notamment, il est largement envisageable d'employer les même procédés pour manipuler des primitives visuelles. Le plus difficile dans cela serait d'identifier un ensemble minimal de générateurs de masse visuelle (c'est à dire ce qui donne l'existence à un objet visuel), ainsi que d'opérations sur les formes géométriques (en deux ou trois dimensions), de façon à obtenir également sur ce plan une expressivité totale. Moyennant cela, le modèle permettrait de décrire des partitions audiovisuelles, ce qui constituerait un avantage certain.

2.1.2 Implémentation

La réalisation concrète du modèle n'en est encore qu'au stade expérimental, étant donné qu'elle n'a jusque là été utilisée que par nous-mêmes, dans des conditions qui permettaient d'adapter le logiciel suivant nos besoins contextuels. Certaines fonctionnalités non décrites ici ont d'ailleurs été ajoutées pour simplifier notre travail.

L'outil fonctionne relativement bien, et d'une façon relativement stable, et nous estimons qu'il serait dommage de le laisser tomber en désuétude. Nous avons plusieurs pistes pour en assurer la continuation.

Améliorations :

Certains aspects de l'implémentation présentent des limites ou des défauts, qui ne seraient pas acceptables dans le contexte où le logiciel serait distribué plus largement. Un certain nombre de choses mériteraient d'être faites, à supposer que nous en trouvions le temps :

- La gestion des erreurs est encore imparfaite, dans la mesure où le système échoue parfois à identifier une faute sémantique, ce qui produit des erreurs à bas niveau incompréhensibles pour l'utilisateur. La plupart du temps, les erreurs sont correctement identifiées comme des incohérences, mais lorsque le programme est composé de plusieurs fichiers, il est impossible de savoir dans quel fichier se trouve l'erreur, ce qui rend les numéros de ligne difficiles à interpréter. En bref, cet aspect mériterait un peu de travail.
- La documentation existe, mais elle mérite d'être remise à jour, développée, et traduite en Anglais. Cela serait certainement une condition indispensable pour que se développe une communauté autour du langage. Et là encore, c'est une chose qui prend du temps.
- Le système de bibliothèque pourrait être amélioré, en transformant l'ancien fichier SC en une bibliothèque native FLSC par le biais du système de paquetage qui a été ajouté par la suite. Cela semble peut-être un point de détail, mais la maintenance de la bibliothèque serait certainement fortement simplifiée. De plus il serait envisageable de proposer un système permettant de la précompiler, ce qui améliorerait fortement le temps de démarrage.
- Finalement, une interface graphique proposant un interpréteur, la possiblilité de charger des fichiers, de lancer l'exécution, de sauvegarder les enregistrements résultants, etc... rendrait peut-être le programme moins difficile d'accès. Il doit être possible de proposer ce type de fonctionnalité par exemple sous la forme d'un mode Emacs.

Intégration :

Nous avons entre nous évoqué la possibilité d'intégrer FLSC sous la forme d'un greffon pour OSSIA/Score [4], ce qui permettrait potentiellement d'employer des objets synthétisés par FLSC dans des partitions interactives Score. A l'heure actuelle, FLSC n'est que peu réactif et il pourrait être difficile d'effectuer la synthèse hors temps réel dans des délais raisonnables, néanmoins les améliorations possibles du modèle de temporalité peuvent rendre cette intégration plus plausible. En fait, à terme FLSC pourrait être employé pour écrire des éléments de partition Score de façon dynamique, dans la mesure où il pourrait manipuler des structures de temps non déterministes. Cette perspective est relativement lointaine, mais toutefois intéressante.

Réimplémentation :

Comme nous l'avons signalé en introduction, le choix de SuperCollider comme langage d'implémentation a été effectué en quelque sorte par défaut, étant donné qu'il s'agissait du langage le plus facilement disponible pour réaliser une implémentation expérimentale. Cela impose néanmoins la contrainte que FLSC dépend de SuperCollider, et qu'il est impossible de l'utiliser indépendamment (d'ailleurs il devrait se nommer SC-FLSC, pour cette raison).

Nous envisageons donc de procéder à une réimplémentation complète de FLSC, à partir d'autres langages : il pourrait être formulé comme un dialecte d'un langage fonctionnel existant (comme OpenMusic [2]), *via* un paquetage permettant la manipulation selon les principes de l'algèbre fonctionnelle d'un langage ou d'une bibliothèque dédiée à la synthèse sonore.

Il est possible dans ce cas que le code résultant soit beaucoup plus réduit, et puisse bénéficier de l'expertise accumulée dans les langages d'implémentation, les problématiques desquels nous ne sommes pas spécialistes. Dans ce cas, potentiellement, les performances deviendraient aussi bonnes qu'on puisse l'espérer de la part d'un langage destiné à être employé sur le terrain.

2.2 Système de synthèse

L'application la plus aboutie à ce jour du langage FLSC réside dans le système de synthèse Schaefferienne que nous avons produit au cours de ce travail. Celui-ci nous a permis de formuler les expériences que nous avons proposées, mais il constitue par ailleurs un système doué d'un potentiel expressif certain, qui mérite d'être amélioré et mis à disposition de toute personne désireuse d'en faire usage.

2.2.1 Application de terrain

L'analyse de la théorie Schaefferienne a donné lieu à la production d'un synthétiseur, destiné à être le moteur du jeu Sonify. Les premiers essais montrent qu'effectivement les sons produits sont dignes d'intérêt, et qu'il vaudrait peut être la peine de le réécrire sous la forme d'un programme autonome, sans toute la complexité de l'algèbre fonctionnelle qui n'est pas nécessaire dans ce cadre restreint.

Mise à disposition :

Un projet porteur serait de proposer ce synthétiseur à un public de musiciens ou de sound designers, sous la forme d'un plugin VST ou LV2 ; il pourrait constituer un instrument virtuel interessant, bien qu'en réalité il soit plutôt de l'ordre d'un proto-langage, du fait du caractère non borné de son espace de paramètres.

A supposer qu'il soit possible d'adapter son paramétrage aux interfaces logicielles standard, les algorithmes de synthèse sous-jacents peuvent quant à eux être réalisés au moyen d'un langage tel que FAUST [22] ; ainsi, l'univers sonore auquel il donne accès pourrait de façon bien plus aisée être réapproprié par la communauté des créateurs sonores.

Représentation graphique :

Un logiciel implémentant le synthétiseur pourrait également, au passage, intégrer une interface graphique effectuant la représentation des objets manipulés, et d'une éventuelle partition ; dans ce cas l'objectif aujourd'hui lointain de proposer un outil pour la composition deviendrait sensiblement plus réel. Les éléments graphiques représentant les composantes d'un objet peuvent être empruntés à la théorie de Jean-Louis Di Santo, qui depuis longtemps envisage un système de notation de ce type. Le bénéfice de pouvoir, au moyen de l'ordinateur, mettre directement en parallèle la représentation visuelle et l'interprétation sonore de l'objet, serait d'une aide précieuse notamment dans l'enseignement de la musique concrète.

2.2.2 Extensions du système

Le système est conçu pour être extensible, étant formulé dans un langage qui permet une démarche de développement logiciel. A l'heure actuelle, il forme un tout mais il possède certaines limites qui pourraient être transcendées, à condition d'effectuer les recherches et le travail nécessaire.

Extension des fonctions :

La liste de fonctions modificatrices applicables à un objet est déjà suffisamment longue pour produire des résultats très divers, en particulier puisqu'elles peuvent le plus souvent être combinées à loisir. Néanmoins il est probable que nous sommes loin d'avoir épuisé l'ensemble des fonctions analytiques pertinentes, qu'il s'agisse

de représenter la structure ou l'enveloppe spectrale. Il est très facile, moyennant une équation, de l'intégrer dans l'existant ; le formalisme et son implémentation sont réellement conçus dans un esprit de modularité. En conséquence, poursuivre l'accumulation de cette connaissance présente un interêt certain.

Schémas structurels :

A l'heure actuelle, le système de synthèse ne se préoccupe que d'objets individuels et jamais des structures constitutives d'une partition. Bien sûr, le langage FLSC dans lequel il est implémenté permet d'assembler et manipuler ces objets avec toute l'expressivité d'un langage de programmation, mais nous ne disposons pas d'un modèle indiquant quelles pourraient être les structures pertinentes en ce qui concerne la structuration musicale.

Il est probable que ce sujet peut difficilement être traité d'une façon analogue, puisque les constructions musicales, quant à elles, font l'objet d'une réflexion approfondie depuis un temps bien plus grand ; et au regard de l'Histoire, il semble que les structures pertinentes dépendent fortement de la culture et de la tradition musicale des individus.

Néanmoins, proposer des outils de base pour produire des rythmes, des échelles, des superpositions, et toutes les structures qui semblent être présentes dans toutes les musiques, indépendamment de la culture, pourrait sensiblement améliorer l'intérêt du système, mettant en relation les critères sonores et concrets avec les critères structurels abstraits.

Objets non synchrones :

Le modèle d'objet que nous proposons possède une limitation assez forte, qui n'est pas forcément apparente à première vue, puisque c'est également le cas dans l'immense majorité des systèmes de synthèse que nous avons pu rencontrer : les objets sont *a priori* supposés *synchrones*, c'est à dire que toutes leurs composantes ont la même durée de vie et se situent dans la même temporalité.

Bien que cette supposition, au regard de la façon dont le son est produit dans la nature, semble être simplement une question de bon sens, nous ne devons pas préjuger qu'un objet ne présentant pas cette propriété serait nécessairement dissocié – c'est à dire que ce ne serait pas un objet. Au contraire, il reste concevable qu'il existe des objets *non synchrones*, constitués de composantes indépendantes temporellement, mais qui produisent tout de même une impression d'unité.

Afin de vérifier cette hypothèse, il serait donc nécessaire d'ajouter dans le système la possibilité de produire de tels objets ; par exemple en pouvant spécifier *plusieurs* temporalités, et en affectant chaque composante à l'une d'entre elles, au moyen d'une nouvelle série de fonctions modificatrices.

Modélisation d'autres systèmes :

Puisque la modélisation de la théorie Schaefferienne s'avère fructueuse, nous pouvons alors envisager de reprendre cette opération avec d'autres systèmes musicologiques ; par exemple, la théorie des Unités Sémiotiques Temporelles [20] constituerait un bon candidat. Et dans l'éventualité où cela serait possible, nous pourrions alors envisager un comparatif entre les deux systèmes, dans l'espoir de pouvoir effectuer une synthèse reprenant les points forts et les complémentarités des systèmes d'origine.

Cette démarche, répétée autant que possible, pourrait finalement conduire à l'émergence d'un meta-système combinant les nuances d'un grand nombre de sources distinctes ; bien entendu, cela reste très hypothétique et seul le temps pourra nous dire ce qu'il en est.

2.3 Démarche participative

La tentative en cours pour obtenir la participation du public à la recherche musicale semble constituer une approche très prometteuse, autant dans la façon dont elle alimente le processus en ressources humaines, ce qui est une des limites les plus dures, que dans le fait qu'elle produit des résultats issus d'une émergence sociale qui favorise la transmission des connaissances et des savoir-faire.

Cette démarche novatrice nous semble intéressante à poursuivre, dans la mesure où elle propose de nouveaux modes de questionnement, qui apporteront peut-être de nouvelles réponses aux problèmes rencontrés dans l'évolution de la culture musicale contemporaine.

Là encore, nous pouvons suggérer quelques pistes qui permettront au travers de cette phase expérimentale de développer cette attitude de recherche.

2.3.1 Poursuite des expériences

Il va sans dire que la méthode que nous avons employée jusque là, si elle peut être perfectionnée, peut être employée à nouveau dans l'étude des critères perceptifs – le dialogue avec un expert permet d'identifier des hypothèses plausibles de critères de synthèse pertinents, qui peuvent ensuite être proposés au grand public dans le but d'évaluer leur potentiel à se l'appoprier.

A supposer que le projet Sonify puisse être pérennisé, il suivra le cours qui était prévu dès son origine, c'est à dire que les informations de classification produites par les utilisateurs pourront être analysées afin de tenter d'identifier des schémas récurrents dans l'association entre un algorithme et une catégorie de la perception.

2.3.2 Plateforme de sons libres

Nous pensons également qu'une telle démarche de participation du public pourrait bénéficier d'être plus spécifiquement destinée à une couche de la population présentant un intérêt particulier pour le son, notamment des musiciens ou des sound designers, qu'ils soient professionnels ou amateurs.

Afin d'obtenir de l'information en provenance d'une masse de personnes capables d'écoute et d'appréciation esthétique des sons, nous avons la possibilité de mettre à leur disposition une grande quantité de sons partageables et réutilisables, notamment au moyen des licences libres. Une plateforme Web dediée pourrait suffire à distribuer les sons très largement, et ne serait-ce que l'information produite par les choix de téléchargement [2] nous renseignerait sur les intérêts des uns et des autres. De plus, il serait possible de proposer, en ligne, un système de classement permettant d'obtenir des recommandations sur des sons similaires. Le système serait finalement dans les mêmes lignes que Sonify, mais sans l'aspect ludique, et plutôt orienté vers la pratique musicale.

Il peut sembler difficile du point de vue des ressources d'héberger une telle base de sons, mais il faut garder en mémoire que le code source définissant un son est quant à lui très compact, et que c'est la seule information qu'il est nécessaire de conserver tant que personne ne demande à écouter ce son en particulier. En fait, il est probable que la base contiendra une majorité de sons qui ne seront jamais entendus, et qu'il n'est donc pas intéressant de synthétiser. Et dans l'éventualité où certains sons auraient une certaine popularité au cours d'une certaine période, il serait toujours possible de les garder en cache afin d'accélérer le traitement.

2.3.3 Génération automatique d'une base de sons

Alimenter une telle démarche supposerait de pouvoir produire (ou au moins de pouvoir spécifier) une grande quantité de sons, par exploration des combinaisons accessibles par l'intermédiaire du système de synthèse. Il est envisageable de les produire en combinant des règles de grammaire générative, qui peuvent dans un premier temps rester très simples, mais peuvent également être rafinées afin de générer plus de nuances.

L'explosion combinatoire engendrée ne nous permettra pas d'aller très loin dans l'exploration de cet arbre sans reposer sur des heuristiques nous permettant de chercher dans des directions jugées prioritaires. Le levier sur lequel nous pouvons jouer consisterait à produire un grand nombre de règles de génération décomposant les règles générales en cas particuliers. Alors les choix des utilisateurs permettraient de déterminer quelles règles sont pertinentes en pratique, c'est à dire celles qui sont liées aux intérêts de ceux-ci.

Finalement, une telle démarche relève de l'algorithmique génétique, le facteur de sélection étant fourni par les choix des utilisateurs. Un protocole adapté peut être développé en temps utile, afin de permettre l'émergence de sons générés automatiquement et présentant plus probablement un intérêt particulier pour l'auditeur humain. Il s'agirait donc d'un processus de création distribué de façon éminemment collective.

3 Conclusion

A l'issue de ce processus de recherche, il semble que la question d'une notation musicale symbolique incluant les critères du matériau sonore n'est toujours pas résolue, et qu'elle ne le sera peut-être jamais...

En effet, au travers des approches envisagées, la difficulté apparaît sous diverses formes, mais reste toujours écrasante ; qu'il s'agisse de l'opacité des langages informatiques suffisamment expressifs, du labyrinthe combinatoire des fonctions pouvant modéliser le matériau, ou de la charge de travail colossale que représente l'expérimentation ; la complexité inhérente au sujet, l'incompatibilité des théories, la diversité des points de vue, tout ceci manifeste l'impossibilité pratique de synthétiser la nature de la perception musicale.

Et en fait, une question apparaît alors comme indispensable : est-il réellement nécessaire de résoudre cette question ? Vouloir disposer d'un moyen définitif de complètement représenter la musique, dans ses moindres

2. Une solution pour améliorer la qualité de cette information serait de proposer les sons à l'écoute préalablement au téléchargement, afin d'effectuer un tri entre les choix positifs et les tentatives infructueuses.

détails, et ce du point de vue d'une théorie analytique qui ne laisse pas la place à l'ambiguïté de l'interprétation, n'est-ce pas tenter de figer une bonne fois pour toute sur le papier une chose qui dans son essence est aérienne, comme le dit Dolphy, qui va et vient, et ne se fixe jamais ?

D'un point de vue plus pragmatique, l'importance de la culture dans l'appréciation de la musique est un phénomène incontournable ; or la culture est une chose vivante, en changement perpétuel ; il n'y a pas de culture aboutie. Plutôt que de chercher un système de notation qui n'aurait de valeur que dans un certain contexte culturel, cherchons plutôt les moyens de créer sans cesse de nouvelles interprétations, de nouvelles grilles de lecture qui permettront de questionner, de critiquer et d'analyser des musiques que nous ne connaissons pas encore ; en musicologie, on parle d'*analyse paradigmatique* lorsqu'on tente en premier lieu de saisir le sens d'une œuvre en référence au contexte intellectuel de son créateur, et non par rapport à notre propre système de pensée.

Finalement donc, peut-être que cette question est une question sans réponse, car elle appelle à chaque fois de nouvelles questions, qu'elle réclame de nouveaux points de vue, qu'elle nous pousse invariablement à révolutionner notre mode de pensée afin de saisir la musique dans ce qu'elle a d'unique et de singulier... et si un ordinateur pouvait s'interroger à notre place, et nous dire quel est le sens de la musique, serait-ce encore pour nous de la musique ?

*

* *

Je souhaiterais conclure ce manuscrit avec quelques remarques personnelles :

Au cours de ces quatre années de travail en thèse de doctorat, et au travers des nombreux contextes intellectuels et relationnels que j'ai rencontré, il me semble avoir énormément appris ; non seulement en termes de connaissances que j'ai pu découvrir, ou de compétences que j'ai pu développer, mais également dans mon regard sur la démarche de recherche elle-même, et dans la façon dont je me permets maintenant de questionner la façon dont nous créons chaque jour de nouvelles vérités.

J'ai également appris une chose fondamentale : que finalement tout notre savoir accumulé n'est qu'un aveuglement, si nous ne sommes pas capables d'aborder une question nouvelle avec le sentiment distinct que nous n'en connaissons pas d'emblée la réponse. Conserver la capacité à chercher sans savoir est un élément essentiel dans le travail du chercheur.

En outre, j'ai découvert que ma motivation principale en tant que chercheur n'était pas, comme je le croyais au départ, de *trouver* des réponses à des questions ; bien au contraire, j'ai constaté qu'une question dont on a déjà la réponse cesse rapidement d'être une question intéressante. Finalement, ce qui fait l'intérêt de cette activité de recherche, et qui en est le moteur, c'est de *chercher* des réponses, au risque de n'en trouver aucune, ou parfois de répondre accidentellement à une autre question... Ce questionnement incessant, bien qu'étant parfois épuisant, devient une seconde nature, qui nous alimente et nous fait progresser jour après jour.

Annexe A

Manuel de référence FLSC

FLSC (pour *Functional Language for Sound Composition*) est un langage de type fonctionnel (comme LISP ou SCHEME) ; il est implémenté sous la forme d'une extension de SuperCollider, ce qui permet de réaliser des processus de synthèse sonore avec une grande efficacité.

Le principe général est de spécifier une partition virtuelle (incluant des descriptions d'instruments virtuels) au moyen d'une expression fonctionnelle, puis d'en effectuer de façon automatique la transcription en couple partition-instruments SuperCollider ; après quoi il est possible d'effectuer un rendu, soit en temps réel, soit dans un fichier audio, de la façon usuelle dans ce langage.

Puisque FLSC répond à un problème purement descriptif, c'est un langage sans effet de bord (mis a part bien évidemment la production des artefacts audionumériques résultant des calculs) ; ainsi le sens des expressions ne dépend que des variables libres qui y figurent, ce qui simplifie grandement l'interprétation et la réutilisation du code.

1 Introduction aux concepts fondamentaux de FLSC

On effectue ici un rapide tour d'horizon des principes de fonctionnement du langage. Un ensemble d'exemples est disponible dans le dépôt Git qui hébèrge l'extension SuperCollider associée.

1.1 Types spécifiques FLSC

Outre les valeurs numériques, booléennes, et les fonctions standard, FLSC propose un ensemble de types spécifiques pour la synthèse sonore :

Fragment : la représentation d'un objet audionumérique, qui est produite par un appel à un *instrument virtuel* ou par une combinaison de fragments. Le programme tout entier résulte (en règle générale) en un fragment.

Instrument : un type spécifique de fonction, qui lors de l'appel produit un fragment par combinaison de *signaux*.

Signal : un objet dont la valeur varie au cours du temps, sur un intervalle de temps donné. Les signaux sont produits par appel à des *modules*, et peuvent suivant leur fréquence d'échantillonnage être de type *audio* (si ils sont destinés à être audibles) ou *contrôle* (si ils sont destinés à être passés en argument à des modules).

Module : un type spécifique de fonction, dont le but est de produire des signaux. Ceci est accompli par la combinaison des paramètres (nombres ou signaux) au moyens d'un graphe de *primitives* et d'opérations numériques sur celles-ci

Primitive : une fonction qui retourne un objet dont la valeur varie au cours du temps, suivant les paramètres d'instance (qui sont soit des constantes, soit des paramètres de module, soit d'autres primitives, soit le résultat d'opérations numériques sur l'un de ces types d'objet)

1.2 Structure générale des programmes

Un programme FLSC a pour but principal de spécifier une partition virtuelle (même si il est possible d'effectuer de simples calculs numériques).

En conséquence, la structure générale des programmes est relativement standardisée :

1. L'expression toute entière représente un ensemble de *fragments* qui peuvent être situés dans la temporalité globale par des retards (`delay`) et assemblés dans des listes récursives (c'est à dire des arbres)

2. Les fragments sont produits par appel à des *instruments virtuels* (`patch`) ; ceux-ci ne peuvent être imbriqués, donc toute branche de l'expression contient une et une seule forme `patch`
3. Les instruments sont définis à partir d'appels à des *modules* (`module`) qui produisent des *signaux* (de type *audio* ou *contrôle*) ; à l'intérieur d'un instrument, il est possible de les combiner par des moyens que nous verrons ci-dessous (3.3, 3.4)
4. Les modules sont construits à partir de *primitives* (qui représentent des fonctions temporelles), que l'on peut combiner par des opérations numériques

Bien entendu, il est possible d'employer dans ces expressions les structures standard des langages fonctionnels ; par exemple en donnant tout d'abord des définitions (`let`) ou en effectuant des appels à des fonctions arbitraires.

A l'intérieur des définitions de modules, néanmoins, ces usages sont restreints en conséquence de l'*impératif d'invariance algorithmique* (en effet ceux-ci doivent être précompilés pour des raisons d'efficacité, et donc leur comportement ne peut se transformer dynamiquement). Dans ce contexte les opérateurs fonctionnels sont appliqués immédiatement (à la différence des fonctions standard) dans le but de produire un graphe de calcul à partir d'un ensemble de fonctions temporelles (dont les paramètres du module). Leur application ne peut donc pas dépendre de la valeur d'appel des paramètres (ce sont des paramètres formels qui seront subsitués lors des appels).

1.3 Valeurs temporelles

Une instance de module (un signal) a nécessairement un début et une fin, qui définissent ce qu'on nomme son *support temporel*. En outre certains signaux (ceux qui sont définis par des enveloppes) se réfèrent également à des valeurs temporelles internes au support, que l'on nomme *points-clef* ; une succession de points-clef dans un support est nommée *signature temporelle*.

De façon à assurer la cohérence des valeurs temporelles, celles-ci ne sont pas spécifiées directement à l'intérieur des modules, mais produites au moyen de fonctions spécifiques (cf. 3.4) qui effectuent des transformations cumulatives jusqu'au référentiel temporel global de la partition ; on nomme son *contexte temporel* l'ensemble des transformations qui s'appliquent à une sous-expression donnée.

Le comportement temporel des objets FLSC est déterminé par le contexte temporel dans lequel ils apparaissent. Ils sont spécifiés par une *morphologie non quantifiée* (qui fait abstraction de la quantification des valeurs temporelles), et leur temporalité est ensuite déterminée par l'ensemble des opérateurs temporels qui s'appliquent à eux (c'est à dire leur contexte temporel). Ainsi un même objet peut être réutilisé dans différents contextes, et aura dans ce cas des morphologies potentiellement différentes.

1.3.1 Temporalité globale

Le contexte temporel initial est celui de la partition toute entière ; on parle de temps global et non absolu, puisqu'il est relatif à la date de début du rendu. Le contexte global s'ajuste automatiquement à la temporalité des objets qu'il contient ; c'est le plus petit segment de temps qui suffit à contenir l'ensemble des objets de la partition. En outre, un mécanisme permet lors du rendu d'ajouter des marges de silence au début et à la fin.

A l'intérieur du contexte global, une opération permet de définir un contexte affecté d'un retard ; de cette façon il est possible de situer les objets dans le temps global, et donc ceux-ci peuvent être exprimés dans leur contexte local, en faisant abstraction de leur situation dans la partition. Les retards sont cumulatifs, il est donc possible de situer les objets dans une organisation hiérarchique, à l'intérieur de sous-parties elles-mêmes situées dans des parties d'ordre supérieur, et ainsi de suite jusqu'à la racine de la partition.

1.3.2 Supports temporels

Un support temporel est un contexte temporel local, défini par un début et une fin. L'appel à un instrument virtuel produit un support temporel, dont on peut ensuite dériver des sous-supports inclus. L'hypothèse comme quoi les sous-supports sont inclus dans le support englobant est indispensable à la cohérence temporelle de l'ensemble (et toute exception constitue une erreur).

Un signal audio est toujours défini à l'intérieur d'un support (c'est son domaine de définition), et les signaux de contrôle qu'il prend en arguments doivent être définis sur un support au moins aussi grand. Ainsi on ne peut pas appliquer un sous-support extrait à un signal de contrôle ; l'opération qui le permet doit nécessairement s'appliquer à un signal audio.

1.3.3 Signatures temporelles

Une signature temporelle est une séquence croissante de points-clef à l'intérieur d'un support. Un même support peut donner lieu à une ou plusieurs signatures, ou aucune. Définir une signature ne produit pas d'altération du support, et les signaux peuvent être définis sur des supports signés aussi bien que sur des supports non signés. La signature peut être redéfinie dans un contexte d'ordre inférieur, ce qui écrase simplement la signature précédente.

L'hypothèse comme quoi les signatures doivent être croissantes est l'autre condition indispensable à la cohérence temporelle (et de même toute exception constitue une erreur). Les objets contenant des références à des points-clef doivent nécessairement être définis dans un contexte signé (faute de quoi il s'agit également d'une erreur). Ces objets sont le plus souvent des enveloppes, mais il y a d'autres possibilités.

1.3.4 Découpage suivant une signature

Un opérateur permet d'effectuer le découpage d'un support suivant sa signature, ce qui produit une séquence de sous-supports non signés. Etant donné que cette séquence, une fois assemblée, couvre l'ensemble du support d'origine, il est possible de définir un signal de contrôle comme assemblage d'un tel découpage. Ceci permet notamment de définir une enveloppe comme une série de signaux correspondant chacun à un segment, et ainsi d'obtenir des enveloppes de taille variable [1].

1.3.5 Distribution de support

L'opération inverse est également possible, elle consiste à se référer depuis un contexte à des objets définis dans un contexte plus général. Le signal reçu dans ce cas correspond à une fenêtre prise dans le signal d'origine, sur sa partie commune avec le contexte d'arrivée. La encore il est possible d'appliquer cette opération à des signaux de contrôle.

Un opérateur (cf. 3.1.3) permet de définir *dans le contexte courant* des variables qui pourront être lues dans un sous-contexte, *sans application des transformations successives*. Ceci permet notamment de restituer des phénomènes morphologiques d'ordre général, dans des objets composés de petits éléments.

1.3.6 Temporalité initiale des objets

Avant toute transformation par les opérateurs temporels, la morphologie abstraite des objets est toujours la même :

- Ils sont définis du début à la fin d'un support temporel abstrait
- Dans le cas où ils dépendent d'une signature, leurs points-clef internes sont les indices successifs d'une signature abstraite, à partir de 1

Il est donc nécessaire d'appliquer un support temporel sur n'importe quel objet, et (au préalable) une signature sur les objets à points-clef.

2 Eléments standard des langages fonctionnels

On explique ici dans les grandes lignes comment les fonctionnalités standard sont implémentées en FLSC. Pour une introduction plus détaillée sur les langages fonctionnels, se référer aux sources disponibles (par exemple sur LISP ou SCHEME).

2.1 Syntaxe générale

Les expressions fonctionnelles sont données entre parenthèses. Elles peuvent contenir :

- des *mots-clef* introduisant une forme spéciale
- des *identificateurs* qui désignent soit une variable, soit le nom d'une nouvelle variable (dans les formes spéciales introduisant une fonction)
- des *constantes numériques* dont l'expression régulière est la suivante :
  `(-)?[0-9]+(\.[0-9]+)?(e[0-9]+)?(pi)?`

1. Autrement, les enveloppes en SuperCollider sont nécessairement de taille bornée lors de leur définition, en raison des contraintes qui s'appliquent aux objets du serveur de synthèse.

- le mot-clef `nil` qui désigne la valeur nulle
- des *listes* entre crochets qui contiennent une séquence de sous-expressions

Mis à part dans le cas des formes spéciales, une expression entre parenthèses a le sens d'un appel de fonction, son premier élément est la fonction elle-même et les éléments suivants sont les arguments de l'appel. Les fonctions sont des objets du langage au même titre que les autres et sont donc représentées dans le même espace de variables (tout comme en SCHEME et contrairement à LISP).

Les mots-clef réservés en FLSC sont :

```
lambda   if     let    nowarp   define
 patch   cond   let*     nil    require
module   else  letrec
```

2.2 Formes spéciales

2.2.1 `lambda`

`(lambda (`*name* `...)` *body*`)` retourne une fonction anonyme dont les paramètres sont les *name* `...` ; ceux-ci peuvent être soit de simples identificateurs, soit des identificateurs précédés de `!`, ce qui signifie que la variable introduite peut apparaître plusieurs fois dans le corps de la fonction et qu'en conséquence, si il s'agit d'un élément de synthèse sonore (générateur ou signal), des mesures spécifiques seront mises en place afin d'éviter d'en effectuer des copies identiques lorsqu'il apparaît plusieurs fois dans le même contexte.

En outre le dernier identificateur peut être précédé de `&`, ce qui signifie que la fonction admet un nombre variable d'arguments ; si `N` est le nombre de paramètres de la fonction, les `N-1` premiers arguments seront liés aux `N-1` premiers identificateurs, et tous les arguments restants seront regroupés dans une liste qui sera liée au dernier identificateur.

body est le corps de la fonction, qui sera évalué lorsque celle-ci est appelée.

D'une façon générale, les différents types de fonctions engendrent une clôture lexicale qui mémorise les variables définies dans le contexte dans lequel la fonction est déclarée. Le comportement de la fonction ne dépend donc que de celles-ci, et des paramètres qui sont liés lors de l'appel. Encore une fois, FLSC s'apparente en cela à SCHEME et non à LISP.

2.2.2 `let, let*, letrec`

`(let ((`*name val*`) ...)` *body*`)` est un équivalent sémantique de `((lambda (`*name* `...)` *body*`)` *val* `...)` ; sa réalisation concrète est probablement légèrement plus efficace. Les `name` peuvent être précédés de `!`, par contre le `&` n'a aucun sens dans ce cas.

Tout comme en SCHEME, `let*` effectue la même opération, mis à part que la i$^{\text{ème}}$ *val* peut se référer au j$^{\text{ème}}$ *name*, si j $<$ i.

La forme `letrec` permet de définir des fonctions récursives dans la mesure où les *name* sont ajoutés a posteriori dans la clôture lexicale des *val*, si celles-ci sont des fonctions. Ils ne sont par contre pas présents lorsque celles-ci sont évaluées, on ne peut donc pas définir des valeurs ordinaires de façon récursive (ce qui serait une faute sémantique conduisant à une boucle infinie).

2.2.3 `if, cond`

`(if` *clause then else*`)` évalue l'expression booléenne *clause* ; si celle-ci vaut `true`, alors *then* est évalué et sa valeur retournée ; si *clause* vaut `false`, alors de même avec *else*. Toute autre valeur constitue une faute sémantique (attention, sur ce point le comportement de FLSC est différent de celui de SCHEME). Parmi *then* et *else*, seul celui qui est choisi est évalué ; l'autre a simplement besoin d'être syntaxiquement correct.

`(cond (`*clause consequence*`) ...)` effectue le même type d'opération, mis à part que les *clause* sont évaluées dans l'ordre, et que lorsque l'une d'elles vaut `true`, la *consequence* associée est évaluée et sa valeur retournée. Les autres *consequence* ne sont pas évaluées. La dernière *clause* peut être le mot-clef `else`, ce qui signifie qu'il s'agit de la condition par défaut ; si aucune autre *clause* n'est `true`, alors la valeur est celle de la dernière *consequence*. Comme pour `if`, les *clause* doivent avoir soit `true` soit `false` pour valeur. A la différence de SCHEME, chaque sous-expression doit avoir une et une seule *consequence*, puisque la *clause* choisie vaut toujours true, et qu'il n'y a pas d'effet de bord.

2.2.4 `define, require`

`(define (`*name val*`) ...)` permet de lier à chaque identificateur *name* la valeur quelconque *val*. Dans un programme, l'expression terminale peut être précédée d'un nombre quelconque de `define`, ce qui a pour conséquence que les variables qui y sont liées sont disponibles par la suite. Tout comme pour `let*`, les valeurs peuvent se référer à une variable définie précédemment à l'intérieur du même `define`.

En outre, une séquence de `define` qui n'est pas suivie d'une expression standard constitue un *paquetage*, c'est à dire un ensemble de définitions qui peuvent ensuite être ajoutées dans un autre fichier. Attention, seules les définitions données dans le dernier `define` sont comprises dans le paquetage (mais de toute façon dans aucun cas il n'est nécessaire qu'il y en ait plusieurs).

`(require ''`*filename*`'')` a pour effet d'ajouter dans le programme courant les définitions contenues dans le fichier de paquetage *filename* (exprimé relativement au répertoire du fichier courant). Un nombre quelconque de `require` peuvent apparaître au début d'un programme ou d'un paquetage. En ce qui concerne ces derniers, les définitions des `require` ne sont pas incluses dans le paquetage produit (pour des raisons d'économie), mais elles peuvent y être utilisées et sont sauvegardées dans la clôture lexicale des fonctions qui y sont définies.

2.3 Listes

Les listes en FLSC ont parfois un comportement non standard, car leur rôle n'est pas aussi fondamental que ce qu'il est dans beaucoup de langages fonctionnels (par exemple en LISP, toute expression parenthésée est une liste, qui est évaluée sauf quand elle est échappée par `quote`).

Ici l'implémentation repose sur la notion de *tableau* en SuperCollider ; ceux-ci peuvent être hétérogènes, puisque SuperCollider est un langage orienté objet à typage dynamique (les tableaux contiennent donc des références et non des données). Lorsqu'on effectue une opération sur une liste, la structure de donnée sous-jacente est copiée pour empêcher tout effet de bord, mais cela ne semble pas avoir un impact important sur les performances (il est probable que l'implémentation des tableaux en SuperCollider est fortement optimisée ; par exemple les objets référencés ne sont pas copiés, quant à eux).

En conséquence, la notion de paire pointée n'est pas pertinente en FLSC (on peut accomplir la même chose avec une liste à deux éléments), et il est possible d'accèder sans surcoût à n'importe quel élément d'une liste, ou d'effectuer des ajouts à la fin aussi bien qu'au début. Pour ces raisons la syntaxe des listes en FLSC emprunte à celle des tableaux en LISP – en fait ces notions sont ici confondues.

Etant donné qu'en SuperCollider, les opérateurs numériques sont étendus aux tableaux (ils sont appliqués itérativement aux éléments du tableau, et produisent dans ce cas un tableau), il en va de même pour les listes FLSC. A noter que lorsqu'on applique une opération binaire à deux tableaux de longueur différente, le tableau le plus court est répété de façon cyclique (et lors d'une opération n-aire, le résultat peut dépendre de l'ordre des arguments, qui sont toujours traités de gauche à droite).

3 Eléments spécifiques à la synthèse sonore

On passe ici en revue les formes spéciales et fonctions de bibliothèque qui mettent en place les fonctionnalités fondamentales de FLSC (qu'on a vu en 1). Notamment, leur syntaxe est donnée de façon précise.

3.1 Formes spéciales

3.1.1 `patch`

La forme spéciale `patch` définit un instrument virtuel modulaire ; il s'agit d'un type spécifique de fonction. Sa syntaxe est la suivante :

`(patch (`*parm ...*`)` *pair*`)`

parm ... : une séquence d'identificateurs

pair : une paire `[`*dur signal*`]`

dur : un nombre

signal : un signal

Lors de l'appel à l'instrument virtuel, les paramètres *parm ...* sont substitués, la paire durée-signal `[`*dur signal*`]` est évaluée, et le signal *signal* est situé dans un support temporel local de durée *dur*.

Les paramètres peuvent être précédés de `!` et le dernier paramètre peut être précédé de `&`, tout comme pour `lambda`.

Le signal produit devrait être de type audio, puisqu'il est destiné à être situé comme un fragment audionumérique dans l'espace global de la partition.

Les formes spéciales `patch` ne doivent pas être imbriquées, et ne doivent pas effectuer un appel à un autre `patch`. Si tel est le cas, le comportement est non-spécifié.

Remarque importante : la forme `patch` est désormais obselète, en effet on peut remplacer

`(patch (`*`parm`* `...) [dur sig])`

par

`(lambda (parm ...) (base [0 dur] sig))`

avec une sémantique équivalente.

3.1.2 `module`

La forme spéciale `module` produit un module de synthèse sonore, qui est un type spécifique de fonction.

Sa syntaxe est identique à celle de `lambda`, mis à part quelques restrictions importantes en raison de l'impératif d'invariance algorithmique :

- le nombre d'arguments est nécessairement fixe (la syntaxe `&` n'est pas définie)
- les arguments sont nécéssairement soit des nombres, soit d'autres modules, soit `nil`
- mis à part les générateurs de signaux (primitives et opérations arithmétiques sur les signaux), le corps du module est évalué lors de sa déclaration (et non lors de l'appel) ; notamment les structures de contrôle sont résolues, elles ne peuvent donc pas dépendre des paramètres du module (cela constitue une faute sémantique). Cette évaluation doit produire un graphe de générateurs de signaux.

3.1.3 `nowarp`

La forme spéciale `nowarp` permet d'introduire des variables, tout comme `let` ; sa syntaxe est la même.

La différence réside dans le fait que le contexte temporel qui sera appliqué aux valeurs liées est celui du `nowarp` (et non celui dans lequel elles apparaissent finalement, qui peut contenir des tranformations subséquentes) ; ceci permet aux expressions contenues dans son corps de se référer à des signaux définis sur un support temporel plus général.

La syntaxe `!` peut être employée dans `nowarp`, mais elle est sans effet puisque le comportement de `nowarp` implique déjà que les variables signal ne sont pas dédoublées.

3.2 Primitives des générateurs de signaux

Les primitives peuvent apparaître dans les modules, et en sont les éléments fondamentaux. Elles engendrent des signaux (audio ou de contrôle) dont la valeur peut varier au cours du temps.

Les fonctions numériques peuvent s'appliquer à des signaux ; dans ce cas leur valeur est un signal dont la valeur instantanée est le résultat de l'application de cette fonction aux valeurs instantanées des signaux passés en argument. Dans ce contexte, une constante numérique prend le sens d'un signal constant.

Le type d'un signal est soit fixé par la primitive employée, soit déterminé par les types en entrée ; le plus souvent il s'agit dans ce cas du type minimal qui évite un sous-échantillonnage. Ainsi beaucoup de primitives proposent une variante préfixée par `'hf'`, qui impose un type de sortie audio. Mis à part le cas des générateurs de signaux audibles, ceci permet également de modéliser des phénomènes transitoires dans lesquels l'échantillonnage à fréquence audio des paramètres d'un signal est nécessaire. Dans ce cas, bien qu'il soit techniquement possible de restreindre le support temporel d'un signal de contrôle à haute fréquence, cela ne présente probablement jamais d'interêt.

3.3 Listes de signaux audio

Dans les contextes où un signal audio est attendu (dans le contexte global, à la racine d'un patch, dans une composition), ce signal peut être donné sous la forme d'une liste (éventuellement récursive) dont les éléments terminaux sont des signaux audio, ce qui prend le sens d'une somme de signaux.

Cette règle s'applique également aux arguments de modules, qui ne peuvent pas être autrement des listes ; néanmoins dans tous les cas elle ne s'applique qu'aux signaux audio (la présence de tout autre type d'élément constitue une erreur).

3.4 Altérations de signature temporelle

3.4.1 `delay`

La fonction (`delay` *`theta fragment`*) opère un retard *`theta`* (en secondes) sur un fragment audionumérique *`fragment`* au niveau global. Ceci permet de situer les fragments dans la temporalité globale de la partition.

Etant donné que les fragments peuvent être structurés en listes récursives, il est possible d'organiser la partition en parties et en sous-parties au moyen de plusieurs retards successifs.

Les retards ne peuvent pas apparaître à l'intérieur d'un instrument virtuel.

3.4.2 `sign, base, dur`

(`sign` *`times signal`*) applique au signal *`signal`* une signature temporelle dont les points-clef sont les éléments de la liste de temps *`times`*. Celle-ci doit être donnée en ordre croissant. Le type de *`signal`* est indifférent.

(`base` *`interval signal`*) définit le sous-support *`interval`* (une paire de temps [*`start end`*]) et l'applique au signal audio *`signal`*. Le sous-support doit être inclus dans le support englobant. Le support temporel peut se trouver réduit, et donc *`signal`* doit être de type audio.

Les valeurs de temps peuvent être des nombres, ou des expressions numériques faisant intervenir la variable spéciale `dur`. Celle-ci représente toujours la durée du support temporel local, là où elle apparaît (lier cette valeur à une variable dans un autre contexte n'y change rien, en interne elle représente une fonction).

Les déclarations de supports et de signatures ne peuvent apparaître qu'à l'intérieur d'un instrument virtuel.

3.4.3 `seq, hybrid`

Les opérateurs de composition séqurentielle sont `seq` et `hybrid` (composition hybride, pour les mélanges audio/contrôle).

(`seq` *`signals`*) produit la succession de la liste de signaux *`signals`* sur le support temporel local, découpé suivant la signature temporelle locale. Le résultat de la composition étant nécessairement défini sur le même support temporel, le nombre de signaux doit correspondre au nombre de segments de la signature (c'est à dire le nombre de points-clef, plus un). Effectuer une composition sur un support non signé constitue une erreur.

Les signaux peuvent être de type audio ou contrôle, mais doivent tous être du même type. Il est techniquement possible qu'il y ait moins de signaux que de segments de signature, mais cela risque de produire des comportements imprévisibles (les contrôles sont fixes à leur dernière valeur, l'audio devient nul).

`hybrid` a la même syntaxe que `seq` et effectue la même opération, sauf que les signaux peuvent être de type hétérogène. Le résultat est de type audio.

Les opérateurs de composition ne peuvent apparaître qu'à l'intérieur d'un instrument virtuel.

4 Fonctions de la bibliothèque FLSC

4.1 Fonctions numériques

4.1.1 Constantes

`inf`	l'infini

4.1.2 Opérateurs de signe

`(sgn x)`	signum de `x` (vaut `1` si `x>0`, `-1` si `x<0`, `0` si `x=0`)
`(neg x)`	opposé de `x`
`(abs x)`	valeur absolue de `x`

4.1.3 Opérateurs arithméthiques

`(+ &a)` `(* &a)`	somme et produit des `a`
`(- a &b)` `(/ a &b)`	différence et quotient de `a` par les `b`
`(** a b)`	`a` puissance `b` ; si `a` est un signal non positif, la valeur est égale à `(* (sign a) (** (abs a) b))` (ce qui permet de traiter plus aisément les signaux)
`(mod a b)`	`a` modulo `b`
`(++ a)` `(-- a)`	incrémentation et décrémentation de `a`

4.1.4 Fonctions usuelles

`(cos x)`
`(sin x)`
`(tan x)`
	fonctions trigonométriques

`(log2 x)`	logarithme en base 2
`(floor x)` `(ceil x)`	entier immédiatement inférieur ou supérieur
`(round x y)`	multiple de `y` le plus proche de `x` ; par défaut `y=1`
`(min &a)` `(max &a)`	minimum et maximum des `a`
`(random x)`	valeur pseudo-aléatoire entre `0` et `x` (non-inclus) ; si `x` est entier, alors la valeur est entière ; par défaut `x=1.0`

4.2 Fonctions booléennes

4.2.1 Constantes

`false, fl`	faux
`true, tr`	vrai

4.2.2 Opérateurs booléens

`(not p)`	négation de `p`
`(and &p)` `(or &p)`	conjonction et disjonction des `p`
`(xor &p)`	non-parité des `p` (vrai si un nombre impair de `p` sont vrais)

4.2.3 Comparaisons

`(eq? a &b)`	égalité (vrai si `a` est égal à tous les `b`)
`(neq? a b)`	inégalité (vrai si `a` est différent de `b`)

`(gt? a &b)`
 	
`(lt? a &b)`
	supériorité et infériorité stricte (vrai si les arguments sont strictement décroissants ou croissants)

`(gte? a &b)`
 	
`(lte? a &b)`
	supériorité et infériorité non-stricte (vrai si les arguments sont non-strictement décroissants ou croissants)

4.2.4 Tests

`(? o)` `(nil? o)`	non-nullité et nullité de `o`
`(boolean? o)`	vrai si `o` est booléen
`(number? o)`	vrai si `o` est un nombre
`(function? o)`	vrai si `o` est une fonction
`(list? o)` `(empty? o)`	vrai si `o` est une liste, une liste vide

`(sig? o)`
 	
`(control? o)`
`(audio? o)`
	vrai si `o` est un signal, un signal de contrôle, un signal audio ; `control?` et `audio?` déclenchent une erreur si l'objet n'est pas du type signal

4.3 Listes

Remarque : dans certains cas, lorsqu'une liste est demandée et que l'argument n'en est pas une, il est remplacé par une liste à un élément, ou par la liste vide si il s'agit de `nil`. Notamment dans le cas de `cons`, cela signifie qu'il n'y a pas de paire pointées en FLSC ; en revanche `(cons e nil)` vaut effectivement `[e]`.

4.3.1 Constructeurs

`(list &e)`	liste composée des `e`
`(cons e l)`	ajout de `e` au début ou à la fin de `l` (éventuellement converti en liste)
`(add l e)`	
`(insert l e i)`	ajout de `e` à l'indice `i` dans `l` (éventuellement converti en liste)
`(range a b)`	liste des entiers consécutifs entre `a` et `b` (inclus)
`(range0 n)` `(range1 n)`	liste de `n` entiers consécutifs à partir de `0` ou `1`
`(step n dn n0)`	liste de `n` nombres d'incrément `dn` à partir de `n0`
`(repeat e n)`	liste de `n` occurences de `e`

4.3.2 Accesseurs

`(elt l i)`	élément d'indice `i` dans `l`
`(subseq l i j)`	sous-séquence de `l`, entre les indices `i` et `j` (non-inclus)
`(length l)`	nombre d'éléments de `l`
`(position l e)`	position de la première occurence de `e` dans `l`

4.3.3 Opérations sur les listes

`(append &l)`	concaténation des `l` (éventuellement convertis en listes)
`(reverse l)`	retournement de `l`
`(flop l)`	transposition de la liste de listes `l` (inversion de l'ordre des indices)
`(select f l)`	liste des éléments de `l` qui vérifient la fonction booléenne unaire `f`
`(apply f l)`	appel à la fonction `f` avec les éléments de `l` pour arguments

4.3.4 Itérations

`(mapcar f &l)`	liste résultant des applications de `f` aux i-èmes éléments des listes `l` ; si celles-ci ne sont pas de la même longueur, elles sont répétées cycliquement jusqu'à la longueur la plus grande
`(reduce f l a)`	application incrémentale de la fonction binaire `f` au résultat précédent et aux éléments successifs de `l` ; si `a` est défini, alors c'est la valeur initiale ; sinon la valeur initiale est le premier élément de `l`

4.4 Primitives pour la synthèse sonore

Dans la plupart des cas les paramètres des primitives possèdent des valeurs par défaut et peuvent donc être omis lors de l'appel.

4.4.1 Oscillateur

`(osc freq phi)`	oscillateur sinusoïdal (audio) de fréquence `freq` et de phase initiale `phi` ; par défaut `freq=440` et `phi=0`

4.4.2 Enveloppes

`(seg start end)`
 	
`(hfseg start end)`
	segment linéaire d'enveloppe, variant entre `start` et `end` du début à la fin du support temporel local ; par défaut `start=0` et `end=1`

`(env levels)`
 	
`(hfenv levels)`
	enveloppe linéaire à nombre de points fixe, qui traverse les valeurs de `levels` selon les points-clef de la signature locale, entre le début et la fin du support local ; le support doit être signé, et avoir suffisamment de segments (le dernier segment de l'enveloppe est prolongé sur les segment restants de la signature)

4.4.3 Modulateurs

`(lfo freq phi)`
 	
`(tri freq phi)`
	modulateurs sinusoïdal et triangulaire ; analogues à `osc`, mais pour `tri` la phase est donnée entre `0` et `4`

`(squ freq phi pw)`	modulateur carré ; le paramètre supplémentaire est la largeur d'impulsion sur `[0,1]`, par défaut `0.5` ; la phase est donnée entre `0` et `1`
`(hftri freq phi)` `(hfsqu freq phi pw)`	équivalents haute-fréquence de `tri` et `squ` ; l'équivalent de `lfo` est `osc`

4.4.4 Générateurs pseudo-aléatoires

`(rand max min freq)`	génère des paliers pseudo-aléatoires entre `min` et `max`, à la fréquence `freq` ; par défaut `max=1`, `min=0`, `freq=500`
`(crand max min freq)`	idem avec une ligne brisée (signal continu)

4.4.5 Autres

`(k2a sig)`	convertisseur de contrôle en audio (sous-échantillonné)

4.5 Modules prédéfinis

La bibliothèque contient des modules prédéfinis qui correspondent à l'application élémentaire d'une primitive ou d'une fonction numérique ; pour ces derniers, le nombre de paramètres des modules étant fixe, les opérateurs sont strictement binaires.

On ne donne ici que les correspondances, les paramètres étant les mêmes. Les noms des modules sont les noms des primitives, précédés de `'m'`.

Remarque : `env` ne peut pas être trivialement converti en module, `menv` désigne en fait une fonction que l'on verra plus bas. Il en va de même en ce qui concerne `hfenv`.

4.5.1 Modules correspondant aux primitives

Modules	Primitives
`mosc mlfo mtri mhftri msqu mhfsqu`	`osc lfo tri hftri squ hfsqu`
`mseg mhfseg`	`seg hfseg`
`mrand mcrand`	`rand crand`
`mk2a`	`k2a`

4.5.2 Modules arithmétiques

Modules	Primitives
`m+ m* m- m/ m** mmod` (binaires)	`+ * - / ** mod`
`msgn mneg mabs`	`sgn neg abs`
`msin mcos mtan mlog2`	`sin cos tan log2`
`mfloor mceil mround`	`floor ceil round`
`mmin mmax`	`min max`

4.6 Fonctions génératrices d'enveloppes

Les fonctions génératrices d'enveloppe permettent d'exploiter la structure modulaire pour produire des enveloppes à nombre de points variables à partir de modules `mseg` et `mhfseg`.

`(menv levels)`
 	
`(mhfenv levels)`
	modules parcourant les niveaux `levels` entre le début et la fin du support temporel local, avec des césures sur les points-clef de la signature locale (utilise `seq`)

`(mhybenv levels type)`	idem avec une composition hybride ; le i-ème segment est de type audio si l'indice `i` de type booléen est vrai, et de type contrôle si il est faux (noter que le nombre de segments est la longueur de `levels` moins `1`)
`(lmenv levels)`	fonction engendrant un appel à un module ad-hoc ; les paramètres sont les mêmes que ceux de `env` ; noter que dans ce cas chaque appel produit un nouveau module (ce qui est peu économique)

4.7 Fonctions diverses

`(mn+ &a)`
 	
`(mn* &a)`
	fonctions engendrant une chaîne de longueur arbitraire de modules `m+` ou `m*`, ce qui permet d'obtenir un assemblage n-aire

`(basemap bases sigs)`	fonction permettant d'appliquer à chaque élément de la liste de signaux `sigs` le support temporel d'indice correspondant dans la liste `bases`

5 Exemples

5.1 factorielle.flsc

```
;;;; exemple de calcul simple: la factorielle

;;; on utilise l'arithmétique, une comparaison,
;;; une conditionnelle, une fonction, et un letrec

;; on définit une fonction récursive avec letrec
(letrec ((fact (lambda (n)
                 ;; on compare la valeur avec 1
                 (if (gt? n 1)
                     ;; cas récursif: fact(n) = n * fact(n-1)
                     (* n (fact (- n 1)))
                     ;; cas de base: fact(1) = 1
                     1)))) ; fin des variables
  ;; on calcule fact(6)
  (fact 6))
```

5.2 patch-minimal.flsc

```
;;;; exemple minimaliste de partition
;;;; oscillateur et enveloppe dynamique à deux segments
```

```
;;; on utilise un instrument virtuel [lambda],
;;; un module arithmétique [m*],
;;; le générateur sinusoïdal [mosc],
;;; la signature temporelle [sign],
;;; la composition séquentielle [seq],
;;; et deux segments d'enveloppe [mseg]

;; on déclare et on appelle une fonction sans paramètres
((lambda ()
   ;; la valeur est un signal sur le support [0 durée]
   (base [0 2] ; durée de 2 secondes
     ;; le signal (de type audio) est le produit
     ;; d'un oscillateur et d'une enveloppe
     (m* (mosc 220) ; produit et oscillateur à 220Hz
       ;; enveloppe
       ;; on doit d'abord déclarer la signature temporelle
       (sign [0.5] ; on ajoute un point de césure à 0.5 secondes
         (seq ; seq effectue le découpage du support suivant la signature
           ;; liste des éléments composés séquentiellement
           [(mseg 0 0.1) ; un segment de 0 à 0.1
            (mseg 0.1 0)])))))) ; un segment de 0.1 à 0
```

5.3 menv.flsc

```
;;;; exemple de menv (générateur d'enveloppe)
;;;; la même chose que 02 en utilisant un raccourci

;;; la composition séquentielle [seq]
;;; et les deux segments d'enveloppe [mseg]
;;; sont remplacés par un générateur d'enveloppe [menv]

;; début comme 02
((lambda ()
   (base [0 2]
     (m* (mosc 220)
       ;; enveloppe
       ;; on déclare toujours la signature temporelle
       (sign [0.5]
         ;; menv produit la même enveloppe qu'en 02
         (menv [0 0.1 0])))))) ; l'argument est la liste des niveaux
```

5.4 parametres.flsc

```
;;;; exemple de patch avec parametres
;;;; la même chose que 03 avec une fréquence,
;;;; une amplitude et une durée parametrables

;; on déclare des paramètres
((lambda (amp freq time)
   (base [0 time] ; time est la durée
     (m* (mosc freq) ; freq est la fréquence
       (sign [(/ time 4)] ; on réutilise time pour la césure
         (menv [0 amp 0])))))) ; amp est l'amplitude max
 ;; on appelle avec les mêmes paramètres
 0.1 220 2)
```

5.5 parametre-signal.flsc

```
;;;; exemple de patch avec parametre signal
;;;; la même chose que 04, sauf que la fréquence
;;;; est un signal pseudo-aléatoire

;; l'instrument est le même
((lambda (amp freq time)
   (base [0 time]
     (m* (mosc freq)
       (sign [(/ time 4)] (menv [0 amp 0])))))
 ;; le paramètre de fréquence est un signal
 0.1 (m* 220 (mrand 0.9 1.1)) 2)
```

5.6 generateur-parametre.flsc

```
;;;; exemple de générateur de paramètre
;;;; on introduit une fonction qui calcule une enveloppe

;; introduction de fonction (générateur d'enveloppe)
(let ((adsr (lambda (sus)
              ;; on précise l'allure en faisant abstraction du temps
              (menv [0 1 sus sus 0]))))
;; l'instrument admet une fonction en paramètre
  ((lambda (amp freq time envgen)
     (base [0 time]
       (mn* amp (mosc freq)
         ;; la signature doit avoir le bon nombre d'éléments
         ;; dur est une variable spéciale qui renvoie à la fin du support
         (sign [0.1 0.2 (- dur 0.5)]
           ;; on applique la fonction passée en argument
           (envgen 0.5)))))
   0.1 220 2 adsr))
```

5.7 sous-signature.flsc

```
;;;; exemple de sous-signature

;;; on ajoute une modulation à l'enveloppe de 06
;;; cela implique une enveloppe de modulateur
;;; définie sur l'un des segments d'enveloppe
;;; noter que seule la fonction externe change

;; introduction de fonction (générateur d'enveloppe)
(let ((adsr (lambda (sus)
              ;; on utilise seq puisque les éléments sont hétérogènes
              (seq [(mseg 0 1) (mseg 1 sus) ; deux segments standard
                    ;; le segment avec modulation
                    (m* sus ; produit par la constante
                      ;; 1 + (signal sur [-1, 1] * enveloppe)
                      (m+ 1 (m* (mtri 32)
                                ;; on déclare une nouvelle signature
                                ;; à l'intérieur du segment
                                (sign [(* 0.1 dur) (* 0.9 dur)]
                                  ;; on spécifie l'enveloppe du modulateur
                                  (menv [0 0.5 0.5 0])))))
                    (mseg sus 0)]))))) ; fin seq, lambda, adsr, letlist
  ;; l'instrument est le même
  ((lambda (amp freq time envgen)
     (base [0 time]
       (mn* amp (mosc freq)
```

```
         (sign [0.1 0.2 (- dur 0.5)]
           (envgen 0.5)))))
   0.1 220 2 adsr))
```

5.8 liste-oscillateurs.flsc

```
;;;; exemple de liste de signaux
;;;; la même chose que 04 avec une liste d'oscillateurs

;;; note: les listes ne sont acceptées
;;; que pour les signaux audio

;; le paramètre num est le nombre de composantes
((lambda (amp freq time num)
   (base [0 time]
      ;; la liste est produite par itération
      (mapcar (lambda (i)
                ;; la fréquence varie suivant l'indice
                ;; on utilise * et non m*
                ;; car les arguments sont des nombres
                (m* (mosc (* i freq))
                  (sign [(/ time 4)]
                    ;; l'amplitude est l'inverse de l'indice
                    ;; on obtient donc un dents-de-scie
                    (menv [0 (/ amp i) 0]))))
        ;; les valeurs sont les entiers consécutifs à partir de 1
        (range1 num))))
 0.1 220 2 16)
```

5.9 distorsion-harmonique.flsc

```
;;;; exemple de liste de signaux (2)
;;;; la même chose que 08 avec une distorsion harmonique

;; on introduit la fonction de distorsion
(let ((warp (lambda (i) (** i 0.5)))) ; fonction racine
;; le paramètre dist est la fonction de distorsion
  ((lambda (amp freq time num dist)
     (base [0 time]
       (mapcar (lambda (i)
                 ;; la fréquence varie suivant l'indice
                 ;; modifié par la distorsion
                 (m* (mosc (* (dist i) freq))
                   (sign [(/ time 4)]
                     (menv [0 (/ amp i) 0]))))
         (range1 num))))
   0.1 220 2 16 warp))
```

5.10 profil-de-masse.flsc

```
;;;; exemple de liste de signaux (3)
;;;; la même chose que 09 avec un indice dynamique

;;; on introduit également le split [!] qui permet
;;; d'éviter des clones inutiles

;; on introduit la fonction de distorsion à indice variable
(let ((warp (lambda (i exp) (m** i exp))) ; on utilise un module
```

```
      ;; l'indice dynamique est toujours le même,
      ;; on le multiplie avec split [!]
      (!index (mseg 0.5 1)))
;; le paramètre dist est la fonction de distorsion
  ((lambda (amp freq time num dist)
     (base [0 time]
       ;; split sur l'enveloppe, en sortant la division
       ;; note: le sign doit aller avec, car il définit
       ;; un contexte temporel
       (let ((!dyn (sign [(/ time 4)] (menv [0 amp 0]))))
         (mapcar (lambda (i)
                   ;; ici aussi il faut un module
                   (m* (mosc (m* (dist i) freq))
                       (m/ dyn i)))
           (range1 num)))))
   ;; on obtient la fonction par application partielle
   ;; on augmente le temps pour entendre mieux
   0.1 220 4 16 (lambda (i) (warp i index))))
```

5.11 iteration.flsc

```
;;;; exemple de supports temporels multiples
;;;; la même chose que 08 avec des composantes non-synchrones

;; time est ici la durée de chaque composante
;; dt est le décalage entre les composantes
((lambda (amp freq time dt num)
   (mapcar (lambda (i)
             ;; chaque composante est dans un support différent
             ;; on calcule le début du support
             (let ((start (* i dt)))
               (base [start (+ start time)]
                 ;; on joue sur une gamme à 9 tons
                 (m* (mosc (* (** 2 (/ i 9)) freq))
                   ;; on emploie une enveloppe adsr
                   ;; pour que les composantes soient plus distinctes
                   (sign [0.1 0.2 (- time 0.5)]
                     (menv [0 (* 2 amp) amp amp 0])))))))
     ;; on itère sur les entiers consécutifs à partir de 0
       (range0 num)))
 0.1 220 2 0.5 19)
```

5.12 ordre-superieur.flsc

```
;;;; exemple de supports temporels donnés par une fonction
;;;; le patch est donc une fonction d'ordre supérieur

;; déclaration de la fonction support
(let ((makebase (lambda (i) [(* 0.5 i) (+ 1 i)])))
  ;; patch et application
  ;; ce patch prend en argument une fonction basefunc
  ;; qui lui donne sa temporalité
  ((lambda (amp freq num basefunc)
     ;; la liste des indices
     (let* ((indexes (range0 num))
            ;; la séquence de supports temporel
            (bases (mapcar basefunc indexes)))
       ;; application des supports à un ensemble de signaux
       (basemap bases
```

```
         (mapcar (lambda (i)
                   (m* (mosc (* (** 2 (/ i 5)) freq))
                     ;; la signature est contenue dans le support
                     (sign (mapcar (lambda (t) (* dur t)) [0.1 0.2 0.75])
                       (menv [0 (* 2 amp) amp amp 0]))))
           ;; on itère sur la liste d'indices
           indexes))))
     0.1 220 11 makebase))
```

5.13 nowarp.flsc

```
;;;; exemple de nowarp
;;;; la même chose que 10 avec une fonction de fréquence globale

((lambda (amp basefreq time dt num)
   ;; on calcule la durée totale
   (let ((end (+ (* (-- num) dt) time))
         ;; et également un tableau de hauteurs
         (scale [1 0.75 1.25]))
     (base [0 end]
       ;; on définit une fonction globale de fréquence
       (nowarp ((freq (m* basefreq (m** 2 (mlfo (/ 2 end) 1pi)))))
         (mapcar (lambda (i)
                   (let ((start (* i dt)))
                     (base [start (+ start time)]
                       ;; on module la fréquence globale cycliquement
                       (m* (mosc (m* freq (elt scale (mod i 3))))
                         (sign [0.1 0.2 (- time 0.5)]
                           (menv [0 (* 2 amp) amp amp 0])))))))
           ;; on itère sur les entiers consécutifs à partir de 0
           (range0 num))))))
 0.1 330 2 0.5 13)
```

5.14 nowarp-enveloppe.flsc

```
;;;; exemple de nowarp
;;;; la même chose que 13 avec une enveloppe

((lambda (amp basefreq time dt num)
   ;; on calcule la durée totale
   (let ((end (+ (* (-- num) dt) time))
         ;; et également un tableau de hauteurs
         (scale [1 0.75 1.25]))
     (base [0 end]
       ;; on définit une fonction globale de fréquence
       (nowarp ((freq (m* basefreq (m** 2 (sign [(* 0.5 dur)]
                                           (menv [-1 1 -1])) ))))
         (mapcar (lambda (i)
                   (let ((start (* i dt)))
                     (base [start (+ start time)]
                       ;; on module la fréquence globale cycliquement
                       (m* (mosc (m* freq (elt scale (mod i 3))))
                         (sign [0.1 0.2 (- time 0.5)]
                           (menv [0 (* 2 amp) amp amp 0])))))))
           ;; on itère sur les entiers consécutifs à partir de 0
           (range0 num))))))
 0.1 330 2 0.5 13)
```

Annexe B

Code source du synthétiseur Sonify

Algorithme B.1 Fonction principale – partie 1

```
;;;; synthetiseur Sonify

;;; paquetage requis: interpretation des parametres
;; pour 'gain' et 'haut'
(require "parms.flscpkg")

;;; composition des modificateurs
;; compmass: composition des fonctions de masse
;; liste de [fonction parm ...] -> [num mass auxmass caldyn]
;; le resultat est un tuple de fonctions
(define (compmass (lambda (dates lmass)
                    (reduce (lambda (acc it)
                              (apply (apply (car it)
                                            (append [dates] (cdr it))) acc))
                      lmass
                      [(lambda (n) n)
                       (lambda (i) i)
                       (lambda (i) i)
                       (lambda (i) 1)])))
        ;; mulfunc: produit des fonctions de timbre/couleur
        (mulfunc (lambda (dates lfunc)
                   ;; verifier que la liste est non-vide
                   (if (empty? lfunc)
                       ;; si vide, retourner la fonction unite (neutre)
                       (lambda (freq) 1)
                       ;; sinon
                       ;; calculer la liste des fonctions
                       (let ((funcs (map (lambda (func)
                                           (apply (car func)
                                             (append [dates] (cdr func))))
                                         lfunc)))
                         ;; le resultat est une fonction de la frequence
                         (lambda (freq)
                           ;; effectuer le produit des valeurs de fonction
                           (apply mn* (map (lambda (func) (func freq))
                                           funcs)))))))))
```

Algorithme B.2 Fonction principale – partie 2

```
(define (sonify (lambda (cal time dyn mel lmass lharm lcol)
                ;; calculer les dates, le profil dynamique, le profil melodique
                (let* ((dates (reduce (lambda (acc it)
                                        (add acc (+ (last acc) it))) time [0]))
                       (amp (gain dates dyn))
                       (!freq (haut dates mel)))
                  ;; definir le support temporel
                  (base [0 (last dates)]
                    ;; construire les fonctions modificatrices
                    (let ((fmass (compmass dates lmass))
                          (harm (mulfunc dates (append [[degr [1]]] lharm)))
                          (col (mulfunc dates lcol)))
                      ;; deballer les fonctions de masse
                      (apply (lambda (numfunc mass auxmass caldyn)
                               ;; produit par la dynamique
                               ;; (divisee par le nombre de composantes)
                               (m* (m/ amp (numfunc cal))
                                 ;; iterer sur les composantes
                                 ;; le resultat est une composition parallele
                                 (map (lambda (i)
                                        ;; calcul des frequences
                                        (let* ((frel (mass i))
                                               (auxf (auxmass i))
                                               (fabs (m* freq frel)))
                                          ;; produit par les fonctions de gain
                                          ;; timbre, couleur, calibre dynamique
                                          (mn* (harm auxf) (col fabs) (caldyn i)
                                            ;; oscillateur
                                            (mosc fabs (random 2pi)))))
                                      ;; liste des indices
                                      (range1 (numfunc cal)))))
                             ;; liste des fonctions de masse
                             fmass)))))))
```

Algorithme B.3 Fonctions utilitaires

```
;;;; synthetiseur Sonify -- outils mathematiques

;;; fonction prox
;;; (prox delta x): proximite de x a une valeur entiere (pente delta)
(define (prox (lambda (delta x) (max 0 (- 1 (* delta (fold x 0 0.5)))))))
```

Algorithme B.4 Fonctions de gestion des paramètres

```
;;;; synthetiseur Sonify -- gestion des parametres

;;; fonction 'env': generation d'une enveloppe
(define (env (lambda (dates values)
               (let* ((sel (flop (select (lambda (pair) (? (last pair)))
                                         (flop [dates values])))))
                     (seldates (e0 sel))
                     (selvals (e1 sel)))
                 (sign (cdr (start seldates))
                   (menv selvals)))))
;;; fonctions 'lin', 'mlin': passage de [-1,1] vers [0,1]
        (lin (lambda (x) (/ (++ x) 2)))
        (mlin (module (sig) (/ (++ sig) 2))))

;;; fonctions 'modlin', 'modexp': generation d'un modulateur
;;; fonctions 'parmlin', 'parmexp': generation recursive d'un parametre
(define (parmfuncs (letrec ((modlin (lambda (dates type amt freq)
                                      (m++
                                        (m* (parmlin dates 1 amt)
                                          (m- (mlin
                                                (type (parmexp dates 8 5 freq))) 1)))))
                            (modexp (lambda (dates type amt freq)
                                      (m* (parmlin dates 1 amt)
                                        (type (parmexp dates 8 5 freq)))))
                            (parmlin (lambda (dates maxval parm)
                                       (if (? (car parm))
                                           (apply mn*
                                             (append maxval
                                               (if (number? (car parm))
                                                   (lin (car parm))
                                                   (mlin (env dates (car parm))))
                                               (map (lambda (mod)
                                                      (apply modlin
                                                        (append [dates] mod)))
                                                 (cdr parm))))
                                           0)))
                            (parmexp (lambda (dates baseval oct parm)
                                       (m* baseval
                                         (m** 2
                                           (m* oct
                                             (apply mn+
                                               (append
                                                 (if (? (car parm))
                                                     (if (number? (car parm))
                                                         (car parm)
                                                         (env dates (car parm)))
                                                     0)
                                                 (map (lambda (mod)
                                                        (apply modexp
                                                          (append [dates] mod)))
                                                   (cdr parm)))))))))))
                     [parmlin parmexp])))

;;; recuperation des fonctions primitives parametre
(define (parmlin (e0 parmfuncs))
        (parmexp (e1 parmfuncs)))

;;; fonctions de types de parametres
;; gain: un gain sur [0,1]
;; haut: une hauteur sur 440*2^[-3,3]
(define (gain (lambda (dates parm) (parmlin dates 1 parm)))
        (haut (lambda (dates parm) (parmexp dates 440 3 parm)))
        (intparm (lambda (dates maxval parm) (mround (parmlin dates maxval parm))))
        (parmlin parmlin)
        (parmexp parmexp))
```

Algorithme B.5 Fonctions de masse – partie 1

```
;;;; synthetiseur Sonify -- fonctions de masse

;;; prototype general: fonction de masse
;;; [numfunc mass auxmass caldyn] -> [numfunc mass auxmass caldyn]
;;; numfunc: entier -> nombre positif
;;; mass, auxmass: nombre positif -> frequence relative
;;; caldyn: nombre positif -> gain

;;; paquetages requis
(require "math.flscpkg")
(require "parms.flscpkg")


;;; fonctions de masse
;;; red: redoublement
(define (red (let ((mmass (module (delta i)
                    (+ (ceil (/ i 2)) (* (/ delta 2) (** -1 (mod i 2))))))
                  (mauxmass (module (i) (ceil (/ i 2)))))
              (lambda (dates delta)
                (let ((delta (gain dates delta)))
                  (lambda (numfunc mass auxmass caldyn)
                    [(lambda (n) (* 2 (numfunc n)))
                     (lambda (i) (mmass delta (mass i)))
                     (lambda (i) (mauxmass (auxmass i)))
                     caldyn])))))
;;; brt: bruitage
       (brt (let ((mmass (module (delta i) (+ i (* delta (rand 1 -1))))))
              (lambda (dates delta)
                (let ((delta (gain dates delta)))
                  (lambda (numfunc mass auxmass caldyn)
                    [numfunc
                     (lambda (i) (mmass delta (mass i)))
                     auxmass
                     caldyn])))))
;;; surd: surdensite
       (surd (let* ((order (lambda (smax i)
                             (ceil (log2 (++ (mod (-- i) (** 2 smax)))))))
                    (delta (lambda (smax i)
                             (* (mod (++ (* 2 (mod (-- i) (** 2 smax))))
                                     (** 2 (order smax i)))
                                (** 0.5 (order smax i)))))
                    (mmass1 (module (smax i) (ceil (/ i (** 2 smax)))))
                    (mmass2 (module (smax i) (delta smax i)))
                    (mcaldyn (module (scur smax i)
                               (clip (- (++ scur) (order smax i)) 0 1))))
              (lambda (dates smax scur)
                (let ((scur (parmlin dates smax scur)))
                  (lambda (numfunc mass auxmass caldyn)
                    [(lambda (n) (++ (* (** 2 smax) (-- n))))
                     (lambda (i) (m+ (mass (mmass1 smax i)) (mmass2 smax i)))
                     (lambda (i) (m+ (auxmass (mmass1 smax i)) (mmass2 smax i)))
                     (lambda (i) (m* (caldyn i) (mcaldyn scur smax i)))]))))
       (surd1 (lambda (dates scur) (surd dates 1 scur)))
       (surd2 (lambda (dates scur) (surd dates 2 scur)))
       (surd3 (lambda (dates scur) (surd dates 3 scur)))
;;; cald: calibre dynamique
       (cald (let ((mcald (module (x i) (clip (- (++ x) i) 0 1))))
               (lambda (dates x)
                 (let ((x (parmlin dates 24 x)))
                   (lambda (numfunc mass auxmass caldyn)
                     [numfunc
                      mass
                      auxmass
                      (lambda (i) (m* (caldyn i) (mcald x (auxmass i))))])))))
```

Algorithme B.6 Fonctions de masse – partie 2

```
;;; distquad: distorsion quadratique
;;; distquad2: distorsion quadratique (sans delta)
        (distquad (let ((mmass (module (beta i) (** i (// (++ beta)))))
                        (mcal (module (beta delta i)
                                      (prox delta (** i (// (++ beta))))))))
                  (lambda (dates beta delta)
                    (let ((!beta (parmlin dates 2 beta))
                          (delta (parmlin dates 10 delta)))
                      (lambda (numfunc mass auxmass caldyn)
                        [numfunc
                         (lambda (i) (mmass beta (mass i)))
                         (lambda (i) (mmass beta (auxmass i)))
                         (lambda (i) (m* (caldyn i) (mcal beta delta i)))]))))))
        (distquad2 (let ((mmass (module (beta i) (** i (// (++ beta))))))
                     (lambda (dates beta)
                       (let ((!beta (parmlin dates 2 beta)))
                         (lambda (numfunc mass auxmass caldyn)
                           [numfunc
                            (lambda (i) (mmass beta (mass i)))
                            (lambda (i) (mmass beta (auxmass i)))
                            caldyn])))))
;;; distexp: distorsion exponentielle
        (distexp (let ((mmass (module (beta i) (* i (** 2 (* beta i))))))
                   (lambda (dates beta)
                     (let ((beta (parmlin dates 0.04 beta)))
                       (lambda (numfunc mass auxmass caldyn)
                         [numfunc
                          (lambda (i) (mmass beta (mass i)))
                          auxmass
                          caldyn])))))
;;; decl: decalage
        (decl (let ((mmass (module (delta i) (- i delta))))
                (lambda (dates delta)
                  (let ((delta (gain dates delta)))
                    (lambda (numfunc mass auxmass caldyn)
                      [numfunc
                       (lambda (i) (mmass delta (mass i)))
                       auxmass
                       caldyn]))))))
```

Algorithme B.7 Fonctions de timbre – partie 1

```
;;;; synthetiseur Sonify -- fonctions de timbre harmonique

;;; prototype general: fonction timbre
;;; fonction indice{nombre positif} -> gain{[0,1]}

;;; paquetage de fonctions mathematiques
;;; (prox delta x): proximite de x a une valeur entiere (pente delta)
(require "math.flscpkg")
;;; paquetage de fonctions parametre
(require "parms.flscpkg")

;;; bharm: bruitage harmonique (d'amplitude)
;;; indice de bruitage -> fonction timbre
(define (bharm (let ((mbharm (module (alpha) (- 1 (* alpha (crand 1 -1))))))
                 (lambda (dates alpha) (lambda (x) (mbharm (gain dates alpha)))))))
;;; comb: filtre en peigne
;;; proximite -> fonction timbre
        (comb (let ((mcomb (module (delta x) (prox delta x))))
                (lambda (dates delta)
                  (lambda (x) (mcomb (parmlin dates 10 delta) x)))))
;;; degr: degressivite
;;; ordre de degressivite -> fonction timbre
        (degr (let ((mdegr (module (slp x) (** x (neg slp)))))
                (lambda (dates slp) (lambda (x) (mdegr (gain dates slp) x)))))
;;; dom: dominance
;;; gain, limite -> fonction timbre
        (dom (let ((mdom (module (alpha x0 x) (** alpha (clip (- x0 x) -1 1)))))
               (lambda (dates alpha x0)
                 (lambda (x)
                   (mdom (parmexp dates 2.82842712 1.5 alpha)
                     (parmlin dates 8 x0) x)))))
```

Algorithme B.8 Fonctions de timbre – partie 2

```
;;; presence cyclique de raison 2,3,4
;;; gain, ... -> fonction timbre
        (pcyc2 (let ((mpcyc (module (a2 x)
                              (+ (prox 2 (/ (- x 1) 2)) (* a2 (prox 2 (/ (- x 2) 2)))))))
                 (lambda (dates a2) (lambda (x) (mpcyc (gain dates a2) x)))))
        (pcyc3 (let ((mpcyc (module (a2 a3 x) (+ (prox 3 (/ (- x 1) 3))
                                                 (* a2 (prox 3 (/ (- x 2) 3)))
                                                 (* a3 (prox 3 (/ (- x 3) 3)))))))
                 (lambda (dates a2 a3)
                   (lambda (x) (mpcyc (gain dates a2) (gain dates a3) x)))))
        (pcyc4 (let ((mpcyc (module (a2 a3 a4 x)
                              (+ (prox 4 (/ (- x 1) 4))
                                 (* a2 (prox 4 (/ (- x 2) 4)))
                                 (* a3 (prox 4 (/ (- x 3) 4)))
                                 (* a4 (prox 4 (/ (- x 4) 4)))))))
                 (lambda (dates a2 a3 a4)
                   (lambda (x)
                     (mpcyc (gain dates a2) (gain dates a3) (gain dates a4) x)))))
;;; presence logarithmique d'ordre 1,2
;;; proximite, gain, ... -> fonction timbre
        (plog1 (let ((mplog (module (delta x) (prox delta (log2 x)))))
                 (lambda (dates delta)
                   (lambda (x) (mplog (parmlin dates 10 delta) x)))))
        (plog3 (let ((mplog (module (delta a3 x)
                              (max (prox delta (log2 x))
                                   (* a3 (prox delta (log2 (/ x 3)))))))))
                 (lambda (dates delta a3)
                   (lambda (x)
                     (mplog (parmlin dates 10 delta) (gain dates a3) x)))))

;;; fonction de couleur
;;; gain, freq_basse, freq_haute -> fonction (frequence -> gain)
        (col (let ((mcol (module (alpha f0 f1 f)
                           (** alpha (clip (/ (- (* 2 f) (+ f1 f0)) (- f1 f0))
                                           -1 1)))))
               (lambda (dates alpha f0 f1)
                 (lambda (f) (mcol (parmexp dates 1 1 alpha)
                                   (parmexp dates 880 4 f0)
                                   (parmexp dates 880 4 f1) f))))))
```

Liste des algorithmes

Table des figures

Bibliographie

[1] Abrassart A.-E., *Acoustique de la flûte à bec, Modélisation et expériences d'analyse spectrale*, Mémoire de DEM-E.N.M.D. de Bourg-la-Reine/Sceaux, 2001-2002

[2] Amado C. A. A., *OpenMusic : Un langage visuel pour la composition musicale assistée par ordinateur*, Thèse de Doctorat de l'Université Paris 6, 1998

[3] Camus A., *Réalisation d'un jeu en ligne de classification du sonore*, Mémoire de stage de l'Université de Bordeaux, 2018

[4] Celerier JM., Desainte-Catherine M., Couturier JM., *Extending Dataflows with temporal graphs*, ICMC 2017 (International Computer Music Conference), Shangaï, China, October 2017.

[5] CLM, https://ccrma.stanford.edu/software/clm/, CCRMA.

[6] Church A., *A set of postulates for the foundation of logic*, Annals of Mathematics. Series 2. 33 (2) : 346–366, 1932

[7] Chion M., *Guide des objets sonores, Pierre Schaeffer et la recherche musicale*, Buchet/Chastel & INA/-GRM, 1983

[8] Cohen B., *Analyse du profil harmonique*, mémoire de stage, 2012

[9] Couprie P., *EAnalysis : aide à l'analyse de la musique électroacoustique*, Journées d'Informatique Musicale, Belgique, Mai 2012 : pp.183-189

[10] Dannenberg R. B., *Machine Tongues XIX : Nyquist, a Language for Composition and Sound Synthesis*, Computer Music Journal. 21 (3) : 50, 1997

[11] Di Santo J-L., *Proposition de terminologie structurée pour l'analyse et la composition musicales et de représentation symbolique de la musique électroacoustique champs voisins : sémiotique et linguistique*, Electroacoustic Music Studies, Beijing, 2006

[12] Dybvig R. K., *The Scheme Programming Language, Fourth Edition*, MIT Press, 2009, https://www.scheme.com/tspl4/

[13] GNU, https://fr.wikipedia.org/wiki/GNU_Bison

[14] Hanna P., *Modélisation statistique de sons bruités : étude de la densité spectrale, analyse, transformation musicale et synthèse*, Université de Bordeaux I, 2003

[15] Kelsey R., Clinger W., Rees J. et al., *Revised Report on the Algorithmic Language Scheme*, Higher-Order and Symbolic Computation. 11 (1) : 7–105, 1998

[16] Mc Carthy J., *Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I,* Communications of the ACM, 1960, http://www-formal.stanford.edu/jmc/recursive.html

[17] Meyssonnier T., Desainte-Catherine M., *Sémantique unifiée pour la structuration des objets audio-numériques selon des critères perceptifs, en vue d'une notation pour la musique électroacoustique*, Journées de l'Informatique Musicale, Albi, 2016

[18] Meyssonnier T., https://github.com/tmeysslabri/FLSC

[19] Meyssonnier T., Semal C., http://sndiff.labri.fr/

[20] MIM, *Les Unités Sémiotiques Temporelles - Eléments nouveaux d'analyse musicale,* Eska, Marseille, 1996

[21] Moore B.C.J., *An Introduction to the Psychology of Hearing*, Academic Press

[22] Orlarey Y., Fober D., Letz S., *Demonstration of Faust Signal Processing Language*, Proceedings of the International Computer Music Conference, 2005, Computer Music Association *:* p. 286.

[23] http://opensoundcontrol.org/introduction-osc

[24] Schaeffer P., *Traité des objets musicaux*, Seuil, Paris, 1966/1977

[25] Smalley D., *Spectro-morphology and Structuring Processes*, in Emmerson S. (ed.) *The Language of Electroacoustic Music*, Macmillan, Londres, 1986 : pp.61-93

[26] http://supercollider.github.io/

[27] Shannon C. E., *Communication in the presence of noise*, Proc. Institute of Radio Engineers. 37 (1) : 10–21., 1949, https://fr.wikipedia.org/wiki/Th%C3%A9or%C3%A8me_d'%C3%A9chantillonnage

[28] Thoresen L., *Spectromorphological Analysis of Sound Objects*, Electroacoustic Music Studies, 2006

[29] Varela F., Thompson E., Rosch E., *The Embodied Mind : Cognitive Science and Human Experience*, MIT Press, 1991 (trad. en français par Havelange V. : *L'Inscription corporelle de l'esprit : sciences cognitives et expérience humaine*, Seuil, Paris, 1996).